OS 10.7's Recovery HD: What is it? Why do we need it?

MacTech Magazine
September 2011 • Issue 27.09

MACTECH®

The Journal of Apple Technology

Connect Across Your Universe With KOSMICTASK

Consultant Cowboy: Pricing Ups & Downs

Kool Tools: Adobe Flash Builder & Flex

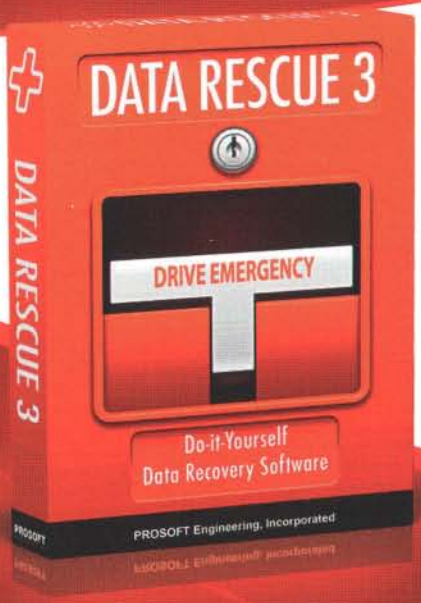
MACTECH.COM

$8.95 US, $12.95 Canada

09

0 321287488 7 8

ISSN 1067-8360 Printed in U.S.A.

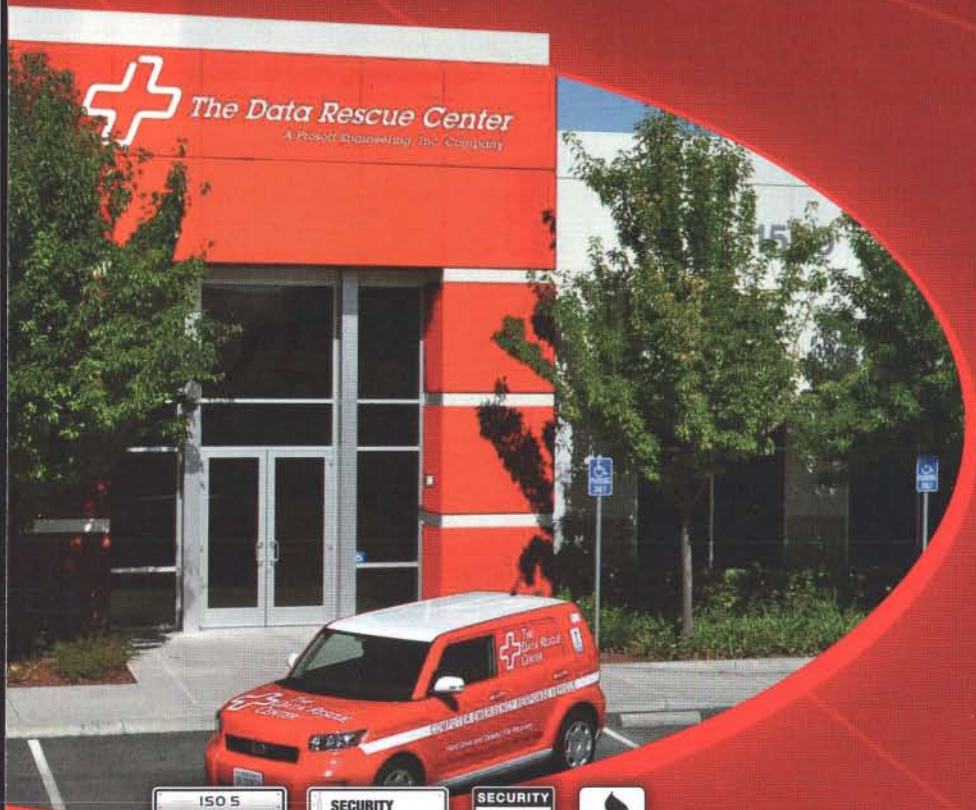


From the makers of...

Data Rescue 3*

The best selling data recovery software for Mac...

...comes the complete data recovery service.



The Data Rescue Center was founded by Prosoft Engineering, makers of the award-winning Data Rescue software. Our recovery know-how provides unparalleled advantages over other recovery services.

- FREE hardware diagnosis
- FREE specialized drive & laptop boxes sent direct to you at NO COST
- No Data, No Charge
- Professional Results at a Lower Cost
- Numerous awards in 2010 for Best Computer Recovery Service
- Class 100/ISO 5 Cleanroom
- Class 2 Vault
- FREE E-Waste Recycling

The Data Rescue Center is headquartered in Livermore, California at a new state-of-the-art facility near Lawrence Livermore National Laboratory.

This new facility provides our recovery engineers with the top technology to recover your computer data while offering best-in-class security features to protect your data from being compromised.

ISO 5 CLASS 100 CLEANROOM | SECURITY UL LISTED | SECURITY | BBB ACCREDITED BUSINESS

diablo BEST OF THE EAST BAY 2010 | The Best of Silicon Valley 2010 metroactive Best Hard Disk Recovery | BEST IN NORTHERN CALIFORNIA Best of the Bay 2010

The Data Rescue Center

hard drive recovery · data migration · photo archiving

1599 Greenville Rd | Livermore CA 94550

877-501-4949

TheDataRescueCenter.com

TABLE OF CONTENTS

From the Editor

We are once again thrilled to bring you this month's issue of MacTech. It's truly an incredible state of affairs that we find ourselves in, and everyone at MacTech is honored to be a part of it. What's going on, you ask?

We all know about Steve Jobs stepping down from his main, day-to-day role at Apple. However, the follow-up is the real story: investor confidence continues and Apple is still, well, Apple. Now, unlike some others, I won't claim it's the *same* company. However, I wouldn't have claimed that had Steve Jobs not gone anywhere. Apple is evolving, and will always continue to do so. The company certainly has built up a culture of disciplined excellence that will carry on.

While the U.S. and the world groans about the economy and jobs in general, technology people are still mostly in demand. Particularly developers. (If you've been reading anything I write, I've been talking about this for *years*.) Apple fuels so much of this. From the mobile space (iOS) to the Desktop (the Mac App Store), the barrier to entry is the lowest it has ever been. Even IT consultants and staff can provide custom-developed solutions that enhance their client's business. Those solutions are often open sourced or sold.

This month's cover story is the work of a developer of a utility for Sys Admins. Great combination, right? Developer Jonathan Mitchell, author of KosmicTask (and now, author in MacTech) describes his utility, what goes into setting it up and using it and its scripting-based architecture.

If you're still wondering how to get started, catch up with Peter Hosey's, "Objective-C: A First Look." Peter started last month with a base level of the C language, and is now describing the foundations of Objective-C. Even if you choose Ruby, Python or other language, for developing on any Apple platform, you'll need to know Cocoa and some Objective-C.

For those that are consulting, or those thinking of doing so, check out this month's Consultant Cowboy, which gets further into the question most consultants wonder about: money. Most of us are competent enough to handle the tech side of things, but the money factors always seem to elude us. Let Ryan Wilcox's experience help you out.

While we have plenty of other important content this month, I'll make mention of one more. In this month's MacTech Spotlight, we feature Hisham Khalifa. Hisham follows his passion for Mac development late nights, after work. If you're in need of some inspiration to get started, start with this month's MacTech Spotlight.

We're a little less than a month out from this year's MacTech Conference. We have great speakers, great talks and great plans for this event. Make sure you're part of it. Check out http://www.mactech.com/conference for more information and registration.

See you next month.

Ed Marczak,
Executive Editor

Communicate With Us

Department E-Mails

Orders, Circulation, & Customer Service
cust_service@mactech.com

Press Releases
press_releases@mactech.com

Ad Sales
adsales@mactech.com

Editorial
editorial@mactech.com
(Authors only, no pr)

Accounting
accounting@mactech.com

Marketing
marketing@mactech.com

General
info@mactech.com

Web Site
http://www.mactech.com

In this electronic age, the art of communication has become both easier and more complicated. Is it any surprise that we prefer **e-mail?**

If you have any questions, feel free to call us at 805/494-9797 or fax us at 805/494-9798.

If you would like a subscription or need customer service, feel free to contact MacTech Magazine Customer Service at 877-MACTECH

We love to hear from you! Please feel free to contact us with any suggestions or questions at any time.

Write to letters@mactech.com or editorial@mactech.com as appropriate.

MACTECH®
The Journal of Macintosh Technology

A publication of XPLAIN CORPORATION

Canada Post: Publications Mail Agreement #41513541
Canada Returns to be sent to: Bleuchip International, P.O. Box 25542, London, ON N6C 6B2

MacTech Magazine (ISSN: 1067-8360 / USPS: 010-227) is published monthly by Xplain Corporation, 705 Lakefield Road, Suite I, Westlake Village, CA 91361. Voice: 805/494-9797, FAX: 805/494-9798. Domestic subscription rates are $47.00 per year. Canadian subscriptions are $59.00 per year. All other international subscriptions are $97.00 per year. Please remit in U.S. funds only. Periodical postage is paid at Thousand Oaks, CA and at additional mailing office.

POSTMASTER: Send address changes to **MacTech Magazine**, P.O. Box 5200, Westlake Village, CA 91359-5200.

by Ryan Wilcox

Pricing and Money: Ups & Downs

Consulting is often "feast or famine" when it comes to money. Here are some tips.

Introduction

The great thing about being a cowboy consultant is that you're in control of your work and it's great when money is coming in like crazy. When you're working so hard you hardly have time to invoice, it's pretty nice knowing that a big payday is on its way.

The flip-side to this is when you can't find work, and your current projects are coming to an end, or your bills are piling up faster than you can pay them off. Sometimes the tips from the "Debugging Pricing Problems" Consultant Cowboy article can help, and sometimes not.

There's often a flow to consultancy, an up and down pattern: busy, slow, busy, slow. The trouble is dealing with the slow times without going broke.

This article will talk about exactly that situation, discussing ways to cope with the "feast then famine" cycle that seems prevalent when being a consultant. The second half of this article has some recommendations on how to avoid an uneven cycle like that in the first place.

Ways to cope

If you're in a slow time, I feel bad for you (son). Been there, done that. Ideally you'd have your primary sources of income and a few backup plans too.

As I've said in the past, most of my work has usually been for client at a time, and these clients always want all of my time. Lately I've transitioned to having several clients who want a day or two of my time, and a few side projects.

One big main client and 1-2 side clients

One way to avoid the ups and downs of contracting work, especially if you have one main client, is to find another client whose work you can do on the side (nights, weekends, whenever). This is a hard one. If you're working 40-50 hours for one client it's often hard to give the other clients the time they need... especially if you have non-billable work to attend to.

Sometimes it's very hard to even find the time to *look* for secondary clients while you're working hard on the main client. However, the disadvantage of having one main client is that when that client discontinues your services, your "feast" time quickly becomes "famine" time. The part time client can provide some security here—at least *some* money is coming in!

Consult with a product on the side

Several very successful .com companies started off as companies doing consultancy work to bring in money while simultaneously developing a product to launch.

Maybe you don't even have a good idea for a product to launch. Software development, at least for me, generates a large amount of byproduct: libraries spun off the main app, applications that could be productized for general use, software used internally to help get the job done better. (For example, writing your own time tracker.)

With the creation of the Mac App Store, I think even small applications make sense as side products now. The Mac App Store reduces the costs of publishing software, especially paid software. No longer do you have to set up the infrastructure for handling downloads and sales for your app: Apple does it all for you (for 30% of the sales price).

I think this changes the dynamics of the software market—assuming you have an idea for a small app, that can be implemented quickly. You can make money on an app that you charge only a few dollars for. A small, side product—one that you can create between client work, or evenings and weekends—might not fully supplement your client income, but it might give you a little cushion for those lean times.

Consulting and a non-computer side business

There's a lot to be said about this idea. For a while I tried to sell books for a profit on Amazon, and one of my former clients also sells customized Swiss Army knives on Ebay, in addition to his main (software) product.

The thing about custom software development, and some IT work, is that percentage wise not many normal people can afford our services. (Granted, this is different if you also offer service and support for customer machines, not just company clients.) With a product, you've potentially opened the market. Who's *not* in the market for a $4.00 Mac App Store app, or a $.99 iPhone app? (Theoretically, anyway)

Blackmagicdesign

Welcome to DaVinci Resolve 8, Hollywood's most powerful color corrector!

With over 25 years' experience in color correction, DaVinci Resolve is the world's most loved high end color grading system! Only DaVinci Resolve is designed to be real time all the time, so it keeps up with you when you're working on demanding client jobs. With the most creative toolset and highest image quality, it's easy to see why DaVinci Resolve is used on more Hollywood feature films, syndicated network television series, music videos and high end television commercials than any other system.

Greater Creativity

With a massive toolset designed by colorists for colorists! The innovative YRGB primaries and node based design allow more creative grading and better looking images. Combined with power windows, RGB mixing, curve grading, blur, sharpen, mist, keying, noise reduction and 32 bit float quality, you get more with DaVinci Resolve.

Super Computer Processing

DaVinci Resolve uses a cluster of GPUs for real time super computer performance. Simply plug in an extra common graphics card (GPU) to get more performance. Add up to 3 GPUs on Mac OS X or a massive 16 GPUs on Linux. The freedom is yours, and there are no extra software costs! Simply plug in GPUs when you need more power!

Automated for Speed

DaVinci Resolve includes more automatic tools such as the 99 point 3D window tracker so you'll rarely need to manually track windows! You get automatic 3D eye matching, auto-grade, auto stabilization, auto 3D color matching, automatic XML, AAF and EDL conforming, real time proxies, auto scene detection and much more!

World's Best Compatibility

No system supports more file formats in real time than DaVinci Resolve. Grade from mixed format clips on the same timeline including raw RED and ARRI, even in bayer format, ProRes™, H.264, uncompressed and more. Get full multi layer timeline XML, AAF and EDL round trip with editing built right into DaVinci Resolve! If your edit changes, Resolve will automatically relink grades!

DaVinci Resolve Lite Free download limited to 1 GPU and 2 nodes, with the same 32 bit float quality.	**Free**
DaVinci Resolve Software Full Resolve wth unlimited nodes and multiple GPUs. Use 3rd party control panels.	**$995**
DaVinci Resolve Full Resolve with colorist control surface for the most advanced facilities.	**$29,995**

Learn more today at **www.blackmagic-design.com/davinciresolve**

Consulting and a highly side automated business

The bad thing about one main client is that sometimes the demands of this client suck the hours out of your week, leaving the side clients starved, giving you no time to work on your products, or taking you away from your workbench and your non-computer side work.

Ideally you'd like a business that could bring in extra income with you putting very little time into it. *The 4 Hour Work Week* by Tim Ferris, is an excellent resource for this kind of business. Some may hate Tim Ferris, or the ideas of this book, but here's where it come into play. Computer programming is a very in demand field, so we can charge very high hourly rates and *still* have people knocking down our door. Any side business that takes time away from this job should (in the ideal work) bring in as much or more money than our hourly rate programming. If it isn't, then why not put in a 44 hour week with your main client instead of putting in 40 hours and spending 4 hours making charm bracelets you're going to sell over the Internet?

But, if you could create a side business that required little or no work by you, you could possibly make—if you do the math—a very good hourly rate on those hours you actually did work.

While the odds may not be all that good to have a mega-successful highly automated side business, we're not looking for that—we're just looking for money to come in to supplement the money we're making in our client work... and maybe cover our rent if a client doesn't pay up, a project doesn't end when we'd like it to, or we're in-between clients.

CD Ladder

Another thing I've debated is setting up a ladder of CD (Certificate of Deposits). Under this idea, I would take an exceptionally good month's profits and divide any extra money up into 4-6 CDs.

For example, if I have a good month, and bring in an extra $2,000, I could divide that into 4 CDs of $500 each. If each CD expires at a different times (3, 4, 5 and 6 months) every month I would have the option of either renewing an expiring CD or letting it roll over for another few months.

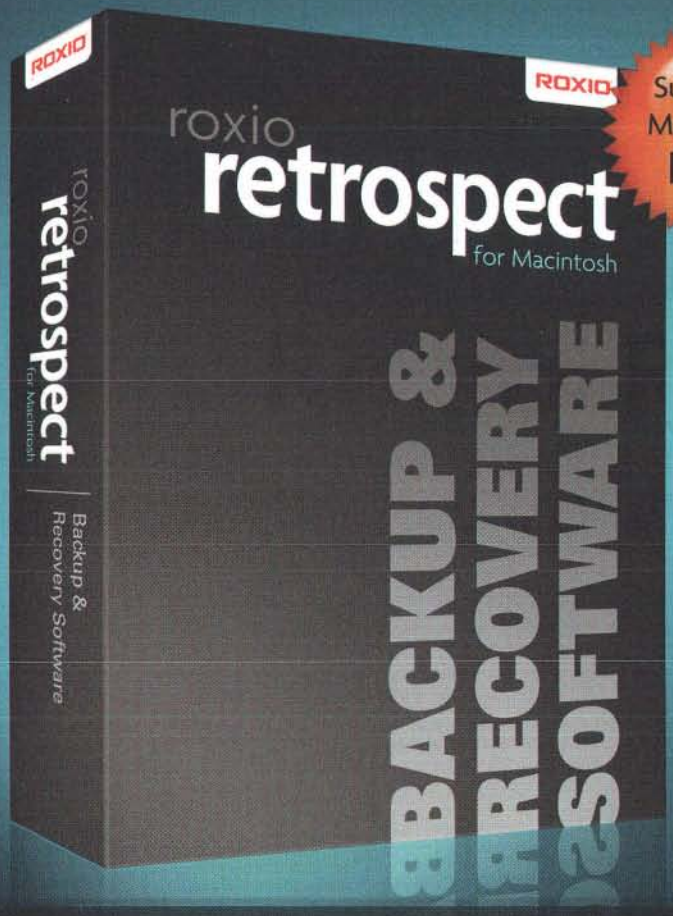
The Simple Dollar weblog has an awesome explanation of putting together a CD ladder (http://www.thesimpledollar.com/2008/10/05/creating-a-cd-ladder-for-your-emergency-fund-or-other-savings-to-earn-a-better-safe-return/).

I like this approach—while it's really easy to tap your savings account for extra money, it's slightly harder to use a CD. This approach also scales up: Have another good month? Just buy CDs at the same interval: you'll have two CDs that expire on the same month instead of one (you could then consolidate those two CDs into one, probably for a better interest rate).

Download a FREE 45-day trial at www.retrospect.com/trial

NEW Roxio® Retrospect® 9

Backup & Recovery for Small Business



Supports Mac OS X Lion

Now With Cloud Storage Support!

NEW IN RETROSPECT 9

- Back up to offsite or cloud storage with new WebDAV capabilities
- Grant on-demand backup and restore permission to Retrospect Clients
- Get great performance on the latest hardware with support for Mac OS X Lion and the latest 64-bit Intel processors
- Use client-side Growl® notifications to communicate when backup activities are running
- Use Retrospect's new S.M.A.R.T. Disc Monitoring feature to be warned of impending hard disk failure so you can take immediate action
- Manage access to end-user features with new administrative controls

To learn more, please contact your local reseller or call 866-825-7694

www.retrospect.com

ROXIO

How to avoid money ups and downs in the first place

Always be Selling

One way to avoid the money up and downs is work very hard—even during busy times—to drum up more work. There are two questions that immediately come to mind

"What happens if I get some of this work I'm looking for? I'm booked, and don't have time to do it now!"

There are two ways I approach the contradiction of finding yet more work when you're *already* up to your eyeballs in work.

If your work tends to be mostly project-based work, then an alternative is to schedule, on the calendar, when you'll be able to get to the new work. For example, if you know your current project ends in two weeks, then schedule this new project to start 20 days from now. I recommend building some slack into your project schedule, as I did here, to avoid issues when a project takes longer to fully implement that you expected (or takes longer to integrate on the client side!)

If your work tends to be long running, "We want a day of your time every week" projects, then those are slightly different.

I spent a good bit of this summer doing networking. This time it was not networking for clients, but networking with other freelancers that might have some availability to do work for me.

This way I can pass off any work that I get, which I don't have time for, to another freelancer. The client is happy because the job gets done, and I'm happy because I didn't have to make a spot in my schedule.

The book *The Unlimited Freelancer* by Hipp and Chartrand (published by the people at http://www.freelancefolder.com) calls this cross-sourcing: when you're hiring freelance talent as good (or better!) than you are, to complete a task that you might not have time to complete yourself.

If you're an advanced web developer, you want to find someone who can complete advanced web development work up to your level. This network can be hard to build, but I'm currently running two projects with some freelance "cross-sourcers", on top of the 3 client projects I'm working on directly.

"I'm working full time with the clients I have—I don't have time to do marketing, I've got paying work I have to do!"

This is tricky, because you're in demand. I would, however, highly recommend taking a day every week and working on your marketing.

You want to do marketing when you're busy to make sure you have jobs coming up to keep you busy. Marketing during one of the down times is not fun (I know, I've been there): you're pressured to find something before money runs out, and

sometimes you back yourself into a situation that might not be ideal for you (too low pay, a project that didn't "feel right", or a deal that will be bad for you in a few months).

Which is why I like building a day of marketing into my budget/schedule, even when I'm super busy. Admittedly, it's very easy to get sucked into client work 5 days a week (at least for me), but 4 days a week of client work is my goal.

Alliances

I think it's just as important to network with other freelancers, as it is to network with potential clients—even other freelancers that don't do the same work as you do.

For example, I've gotten a fair bit of work from marketing agencies who need some help completing a website or computer program for one of their clients. They know that some custom web development has to be done to complete the marketing objective, don't have that experience in house, and hire me to do the required website work.

Maybe a sales consultant might need an IT professional consultant to wire up and install servers at a client location.

Likewise, another thing that happens with freelancers is bidding, or wanting to bid, on a job that's too big for just one person to handle. Taking the job on as part of a team/alliance of freelancers may be the way to succeed.

In addition to providing skills you don't have, or extra hands on a project, an alliance of freelancers may also be a source of leads for you in the lean times, or a place where you could recommend clients when you're simply too busy to take on more work yourself.

Sources of Emergency Funds

When I started Wilcox Development Solutions, my financial advisor started a life insurance fund for me. This life insurance fund was more than just money for my family when I die, but also a fund where I can borrow money when I need it.

Over time this has built up and now I can borrow a fair bit of money from it. I use this as both an emergency fund, and a fund to consolidate some of my loans. For example, the other day I used it to pay off a credit card, essentially refinancing the credit card's interest rate.

I haven't needed to use the life insurance money to pay monthly bills, but it's nice to know that the money is there if I might need it.

The simple savings account shouldn't be ignored here either. January and February (of 2011) were busy months for me. A client had a project they wanted delivered, at all costs, in the middle of February, so we ended up working a lot of hours to see that happen. Being paid by the hour, like I was, meant that I had good months for a few months.

While some of the money I earned went to pay off loans, I also made sure to save some of that money for a rainy day, when there wasn't so much work available.

I have a savings account attached to my business checking account, and I use that as an emergency, or cushion, fund. It's not high tech, and it may not get the best interest rate, but it is convenient.

I could also set up automatic withdrawals from my business checking account to that savings account. I find this especially important if your bank will withdraw from that savings account if you overdraft your checking account: it's too easy to forget to replace the money that the bank automatically transferred to your savings.

Conclusion

I know how hard it is when there seems to be no current projects coming in, and thus no money coming in. Been there a few times myself. Sometimes the trick is to not put too many of your (financial) eggs in one basket, or increasing your network. Sometimes the answer is preparation: being ready for the inevitable.

Sometimes the trick is to reevaluate your pricing (see Debugging Pricing Problems, in Mactech July 2011), and sometimes the trick is to reevaluate your pricing methods: which will be the topic of next column.

Until then, see you, consultant cowboy!

About The Author

Ryan Wilcox has been consulting on his own for the last 8 years, through ups and downs in his business. In 2009 he started thinking about best practices for business, in addition to his normal thinking about programming. He can be found at: http://www.wilcoxd.com. Have thoughts or want to give feedback on this article? rwilcox@wilcoxd.com

Don't get Lost in the Crowd...

...let an Evolis card printer Identify you!

Fully compatible with Mac®, the Pebble[4] ID card printer is the perfect solution to print all your ID cards in high-resolution. Whether you need Employee ID's, Student ID's or Loyalty cards, the Pebble ID card printer will help you stand out from the crowd.

Call TransTech Systems today

1-888-843-3643

to get connected with your local Evolis partner!

Pebble4
Color Single-sided card printer

TRANSTECH SYSTEMS
IDENTIFYING SOLUTIONS

TransTech is an Official Evolis Solutions Provider

Tel: 1.888.843.3643 • **Fax:** 503.682.0166 • **email:** sales@ttsys.com • **www.ttsys.com**

EVOLIS OFFICIAL PARTNER

by Edward Marczak

MacRuby Drag and Drop File Rename

An evolving foundational starter project

Introduction

Recently, we began a MacRuby-based project that demonstrates drag and drop. (Well, strictly, it's just drop that we're showing.) Since it's such a signature feature of OS X and the Macintosh, it's important to understand as drag and drop is deeply ingrained into the user experience. This month, we update the project to operate on the files dropped on to the project. Let's continue.

Installing MacRuby

If you haven't been following earlier articles, we've been using MacRuby, which is a variant of Ruby 1.9. MacRuby is an Apple-backed project that allows the Ruby language to run directly on top of the objective-C runtime. This was covered in the initial article on Ruby in this series, so, for more details, please see that article (MacTech 26.12, December 2010). If you haven't already, download and install the latest version of MacRuby from http://www.macruby.org/downloads.html (version 0.10 as of this writing). MacRuby installs itself separately from the system Ruby, and all binaries are prefixed with 'mac'. (Install MacRuby *after* installing Xcode—ideally Xcode 4.) Please take note of the warning on the downloads page: "[T]his release will only work on Intel 64-bit machines running Snow Leopard (10.6) or higher." You can grab the source and compile up a version that runs under 32-bit, though, if you need to.

The Project

Two articles ago, we put together a complete framework for a drag-and-drop application. As that, it really just was a outline structure that showed us the files that were dropped on the application, but didn't do anything past that. Let's make it do something useful, now. How about an application that checks the filenames of the dropped files and ensures there are no illegal characters? Well, for whatever we define to be "illegal," of course. (File systems differ on illegal characters and it's often a more challenging problem than I have the space to solve here.)

I always encourage you to follow along and create the project along with the article, but, the completed project is available from the MacTech ftp site at ftp://ftp.mactech.com. To follow along, as in past articles, you'll need Xcode. This article will use the latest version of Xcode, Xcode 4. Launch Xcode now, and load the project from last time (or load the completed project you just downloaded and follow along).

Adding Supporting Code

First, we are going to add a new Ruby function to our project. This function will have the sole responsibility of cleaning a filename. We'll keep this function separate, in its own file for two main reasons: reusability and cleanliness. Reusability because who knows where else this code may come in handy? Cleanliness because it makes our job as developer much easier. Since this code isn't directly related to our interface or drag and drop functions, it should stay out of that portion of the code.

To add this new file, choose File->New, or simply press command-N. You'll see the sheet presented in Figure 1.

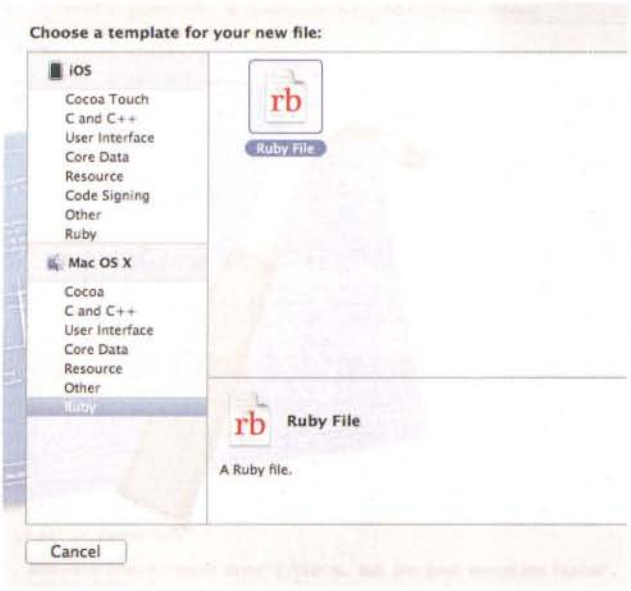


Figure 1 – adding a new Ruby file to the project.

Remember when typing **felt good?**

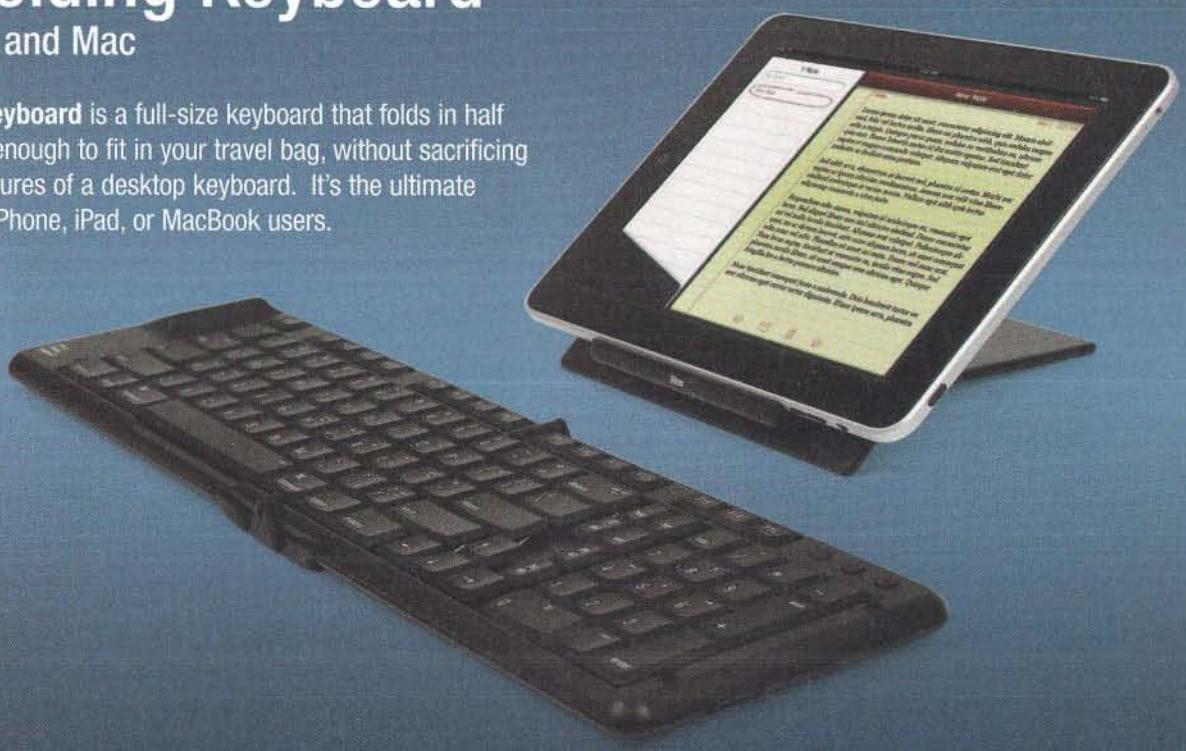
tactile**pro**™

Mac users who crave the **satisfying "click"** of Apple's legendary *Extended* and *Extended II* keyboards will love the **Tactile Pro**. Unlike other keyboards made today, each key is built on a premium Alps mechanical keyswitch. They feel better, and you'll type faster.

Matias **Folding Keyboard**™

for **iPhone, iPad, and Mac**

The **Matias Folding Keyboard** is a full-size keyboard that folds in half for storage. It's small enough to fit in your travel bag, without sacrificing the tactile feel and features of a desktop keyboard. It's the ultimate portable keyboard for iPhone, iPad, or MacBook users.



www.**matias**.ca

Because you've installed MacRuby, you have the additional option to create a Ruby file. Choose that option and click Next, where you'll be presented with the sheet in Figure 2. Save the file as "CleanseFileName.rb".

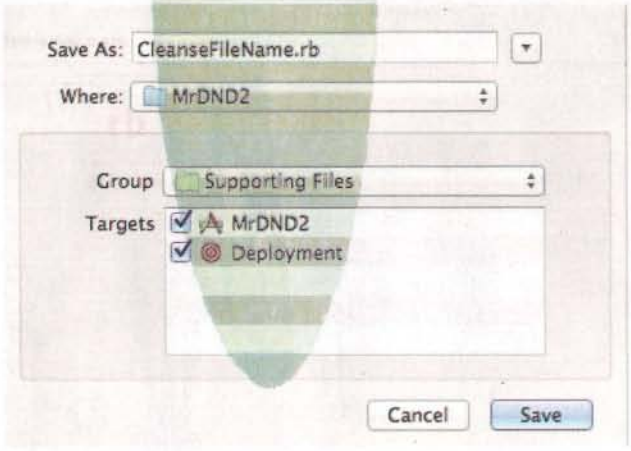


Figure 2 – Adding a file to a folder group and target.

While this file could be placed in any folder group, these exist for your benefit, so use them! Since this is a small project, I'm choosing the "Supporting Files" group. You'd be equally correct in creating an entirely new group and adding the file there.

Check both targets, and click "Save." The contents of the CleanseFileName.rb file is pretty simple. (And again, please note that this is *not* meant to be entirely exhaustive or specific to any one file system.) See Listing 1 for the CleanseFileName function.

Listing 1: CleanseFileName.rb

```
def CleanseFileName(path)
  illegal_map = {
                  " " => "_",
                  "/" => "-",
                  ":" => "-",
                  "*" => "-",
                  "?" => "-",
                  "\"" => "_",
                  "|" => "_",
                  "1" => "1",
                  "e" => "3",
                  "o" => "0",
                  "." => ".",
                }
  filename = File.basename(path)
  dirname = File.dirname(path)
  filename.gsub!(/(.)/) { |c| illegal_map[c] ||
Regexp.escape(c) }

  path = File.join(dirname,filename)
  path
end
```

We have covered all of these data types and loops in past articles. If you're rusty on what they do, take the time to review them now (via those articles or in your favorite Ruby book or resourse). In short, we define a map of characters we consider to be illegal in a file name, and the replacements for each character. We even through in a little l33t speak for fun (and for being obvious while testing). The only trick here is that we map the period character to...a period. This is just a little hack that gets around escaping the period character by the Regexp.escape method.

Back in our AppDelegate method, we need to make some changes. First, add the following require line to the very top of the file, as we're going to depend on the find library:

```
require 'find'
```

Then, update the main loop in the `performDragOperation` function to the code in Listing 2.

Listing 2: Updated loop in performDragOperation

```
for path in paths
  if File.directory?(path)
    # recurse into directory
    Find.find(path) do |f|
      # SetName(f) if not File.directory?(f)
      cleanpath = CleanseFileName(f)
      if f != cleanpath
        File.rename(f, cleanpath)
        NSLog("Renamed #{f} to #{cleanpath}")
      end
    end
  else
    cleanpath = CleanseFileName(path)
    if path != cleanpath
      File.rename(path, cleanpath)
      NSLog("Renamed #{path} to #{cleanpath}")
    end
  end
end
```

Effectively, the only real change is that, unlike the original code which just reported which paths had been dropped on the view, we're calling our new CleanseFileName function. If there's a change to the file name, we touch the file system and rename it (and log the change using NSLog).

In action, we see two offending files in Figure 3:

Figure 3 – Two offending files being dropped on MrDND

Once dropped, you can watch them be renamed as it happens. Don't blink! It happens fast. Figure 4 shows the renamed cleaned files in the Finder.

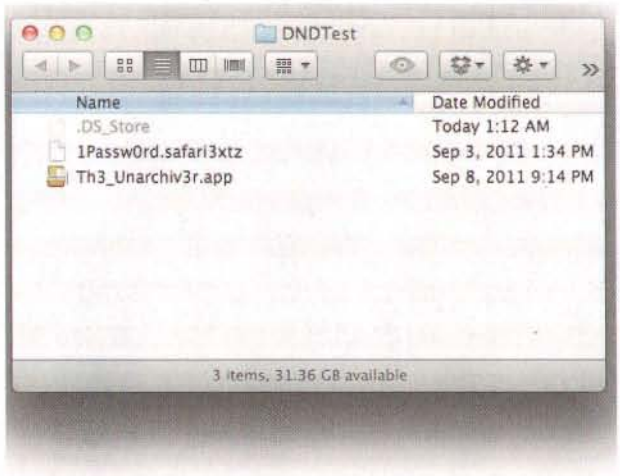


Figure 4 – The newly cleaned files. Note the changes.

Remember that you can drop an entire folder onto the view and it will recurse through the entire structure and rename all offending files found within.

Conclusion

You now have a small application that can accept drag events, get the list of files dropped on a custom view and operate on those files. Congratulations! Of course, there's always room for refinement, and we have plenty to do. While this code works in most normal circumstances, we're not checking for errors and will bomb on files that we don't have permission to rename. Also, we don't give any feedback to the user as to what we're currently doing while renaming. We'll continue to refine and attack these issues next month.

Media of the month: Warzone: Anomoly by 11 bit studios. This small development team from Poland has really hit on a good game. It's a "tower offense" game available for iOS, Mac and PC. I've just been having a great time playing this...you know, in my copious free time!

Until next month, like I've said before, get some more Ruby practice in on your own and don't be afraid to experiment. Hope to see you and discuss MacRuby at MacTech Conference 2011!

About The Author

Edward Marczak is the Executive Editor of MacTech Magazine and writes the monthly "Mac in the Shell" column. He co-founded the MacTech Conference and has authored several books about Macintosh technology. Most recently, this includes* Enterprise Mac Managed Preferences, *published by Apress.

STILL SOLID.
WAY COOLER.

Keep your computers indestructible, free up crucial IT resources, and reduce support costs by 63%.

Faronics Deep Freeze automatically restores workstation configurations with every reboot.

For more information, visit
www.faronics.com

Faronics™

Available for:

Objective-C: a First Look

Building upon C

by Peter Hosey

So you want to write a Cocoa or Cocoa Touch program, and you know C

If you read my article from last month, you know at least enough C to know where Objective-C is coming from. I intentionally skipped a lot of syntax in favor of explaining concepts.

This entry in the series is the same thing, but now we'll be looking at Objective-C's other side: object-oriented programming.

I should note: There are many object-oriented languages, each with different variations on OOP, so don't take this article as defining the One True OOP. There is no right or wrong form of OOP; every language has its own perspective on it. This series is about Objective-C, so the OOP I'm describing here in Objective-C's Smalltalk-influenced flavor.

Objective definitions

Object-oriented programming is where Objective-C gets its name. Objective-C extends C to include features—including new syntax and keywords—to enable OOP.

OOP is exactly what its name says: You think about your program in terms of, and build it from, objects. You build objects by describing them in code, and put them together by describing how they talk to each other.

Objects

If your program is a play, objects are the actors—or, better still, the characters. Each object does some combination of knowing things (state/data) and doing things (methods). Objects communicate with each other by sending each other messages.

Messages

A bunch of characters just standing around and acting separately do not make for a very interesting play. For your program to be at all useful and/or fun, your objects must talk to each other. They do this by sending messages.

Every message has three parts:

- A receiver: The object you are sending the message to.
- A selector: The name of the message you are sending.
- Zero or more arguments.

And does one or more of three things:

- Ask the receiver of the message for some information (usually another object).
- Provide the receiver with some information (usually another object).
- Ask the receiver to do something.

Any code can send a message, but only an object can receive a message—that is, you can only send a message to an object, not to anything else. A C function can send messages to objects, just as objects can, and an object can send itself messages. Objects can also call functions, just as other functions can, but that's not an Objective-C message; it is a function call exactly the same as it would be in a plain C program.

Sending a message to an object is called *message passing*, and it's how you invoke the methods of an object. Methods are very much like C functions, but a method belongs to an object; sending a message to an object is how you call the method's implementation.

One important difference between methods and functions, and between messages and function calls, is dynamic dispatch. A function call is *statically bound*: your function call is hard-wired to that function when you build your executable, so it will only ever call that function, it will jump there directly, and the call will go through every time. A message is *dynamically bound* (dispatched): The method that will run is determined at the time of the message, and no earlier.

Out of this comes another difference, this one between function names and selectors. A C function's name only ever refers to a single function, never to any other function. The function name alone is enough to identify the function, so, in a function call, it is the function name alone that determines which function will be called.

A selector, on the other hand, is not enough by itself to identify a method. It is only part of a message. The other part that determines the method to be called is the receiving object. When you send a message to an object, the Objective-C runtime library looks inside the object to find what method it has for that selector. If it finds one, then it calls that method.

This implies several things:

- Different objects will respond to the message differently. For example, both NSString objects and NSData objects respond

Start a Web Site Today!

from only $4.95/month

Get first month for a penny!
Coupon code: **MACTECH**

Powered by 130% Texas Wind Energy!

100% Mac-compatible Web Hosting

- 99.9% Uptime
- 45-day Money Back Guarantee
- Over 4,500 Free Web Design Templates
- Free Site Builder Software
- In-house Support Available 24x7x365

Sign up at www.HostGator.com/mactech and get the first month for just a penny!

Already have a web site? We'll transfer it for free!

www.HostGator.com **866.96.GATOR**

to `length` messages, but each has its own implementation.

- It is possible not to find a matching method. When this happens, you get an exception that looks like this: "`Object <SomeClass 0xd3c2b1a0> does not respond to selector calibrateFramistan`". This is a symptom of one of several possible bugs in your code.
- It is possible for the object to have no matching method, but provide one upon demand. This is advanced usage that most applications do not have a need for, but it does help demonstrate how dynamic Objective-C can be.

The messages an object responds to are determined by the methods that it possesses, which are provided by the object's class.

Classes

Objects are described by classes. If the objects are characters, their classes are the script. Each class is sort of a template description of an object; when you *instantiate* (create an instance of) a class, the instance is an object that behaves the way you wrote in its class.

In some places, you can treat classes as objects (you can, for example, send messages to them), but most of the objects in a program are instances. You can create as many instances as you need, but a class exists no more than once. Like functions, classes are identified by name alone, so there can't be two classes with the same name.

Of all the things in Objective-C, classes correspond most strongly to modules in C. In fact, classes in Objective-C typically reside in a header file and a module file, each named after the class.

The separation between interface and implementation is even stronger in Objective-C than in C, as Objective-C requires you to declare them in separate sections. As in C, the convention is for the interface to go in the header file and the implementation in the module file.

A class describes itself and its instances. For itself, a class provides *class methods*; when you send a message to a class, a class method is usually what will run. (Yes, usually. The Objective-C documentation goes into more detail; for now, just know that if you want to send a message to a class, you should make sure the class has a class method to respond to it with.)

For a class's instances, it provides *instance methods* and *instance variables*. Just like class methods do for messages to a class, instance methods are what run when you send a message to an instance. (While a message to a class *may* hit a class method, a message to an instance will *always* either hit an instance method or fail.)

Instance variables, also called *ivars* for short, are where an instance keeps state and/or data. Like any other variable, an instance variable is a container that you can put something, such as an object, into. When you declare a variable, you declare its type, which indicates what you can, should, and hopefully will put in it.

A class also usually has a superclass, in which case it is said to "inherit from" or "descend from" that class, and is called a subclass of that class. Everything you declare in a class's interface,

including methods and instance variables, every subclass of that class will inherit. When a class has no superclass, it's called a root class; if you imagine a tree of classes, with subclasses branching off from their superclasses, you can see why.

A subclass can provide its own implementations of methods provided by a superclass; this is called *overriding* the superclass implementation. The subclass implementation can call up to the superclass implementation by sending a message to the special keyword `super`. It doesn't have to be the same message you're responding to (you *can* call the superclass's implementation of a different method), but it's best to only send the same message; doing otherwise will make your code confusing.

In some languages, a class can have more than one superclass; this is called "multiple inheritance". Objective-C does not have multiple inheritance: Each class either has no superclass (is a root class) or has exactly one superclass.

When you use or subclass one of Apple's classes and want to find how to do something in Apple's reference documentation, don't forget to look in the documentation for the class's superclasses. Apple does not redundantly document every inherited method, except *usually* when the subclass changes the method's behavior, so a method you need may actually be implemented by and documented for a superclass of the class you're working with.

Properties

It's quite common for other code to either ask for something from an object or give something to an object. The object enables this by responding to *accessor* messages. A *getter* message returns the current value, whereas a *setter* message changes it. In Cocoa and Cocoa Touch, these should follow a certain naming convention, which is described in Apple's "Coding Guidelines for Cocoa" document.

Properties formalize this in a syntax that explicitly declares a property, meaning a relationship to another object, and not just a pair of methods named according to a certain convention. A property declaration implicitly declares one or two methods (the getter and setter, the latter being optional) and, most of the time, an instance variable in which to hold the value.

The usual way to implement a property is to synthesize it. When you do this, the compiler generates the instance variable, getter method, and (if appropriate) setter method for you. The main alternative is to write any or all of these yourself, which is completely valid, as long as they match the declarations implied by the property. You can even synthesize the property, then implement one or both of the methods yourself; the compiler will implement whatever you didn't.

I'm intentionally skipping a lot of details for now. I won't say much more about properties in this article. Part 3 of this series will show the syntax for declaration and synthesis in brief, and Part 4 will go into much more detail on properties and a couple of related topics.

Protocols

If you're coming from Java or .NET, you'll recognize these under the name "interfaces". (Objective-C uses the term "interface" for the public interface of a class.)

A protocol declares a list of methods and properties. It can declare some as required (the default) and others as optional.

Classes can declare that they conform to one or more protocols. When they do, the methods and properties listed in the protocol implicitly become part of that class's interface; you can expect any class that conforms to a protocol to implement any required methods and properties declared by that protocol.

Subclasses inherit protocol conformance. If a class's superclass conforms to a protocol, then all of its subclasses must also conform to it. This is another thing to look for in superclasses' documentation.

Unlike a class, you can't create an instance of a protocol. A class describes a kind of object (specifically, instances of the class), whereas a protocol describes a specific set of methods a class or its instances must provide. Protocols are most often used to enumerate responsibilities that a class or its instances must fulfill.

A protocol also cannot list instance variables. A protocol lists things that a conforming class must have in its interface; that only makes sense for methods, which are part of the class's interface, not instance variables, which are part of its implementation.

When you declare a class's conformance to a protocol, it isn't necessary to declare everything declared by the protocol again explicitly in the class's interface. Declaring conformance to a protocol means that whatever is required by that protocol is part of the class's interface; no further typing is necessary.

However, you should consider repeating the declarations of any *optional* methods or properties from the protocol that you implement, to make explicit that this class does choose to implement those methods. Whether this is appropriate will depend on the protocol and how you're using it; re-declaring optional items is as optional as implementing them is. One bonus from doing so is that you will get a warning for each optional method you declared but forgot to implement.

Messages vs. methods vs. selectors

A common point of confusion for people new to Objective-C is the difference between a method, a selector, and a message. These are three related, but nonetheless distinct, things.

- A method is the Objective-C analogue of a function. Classes have methods, both for themselves and for their instances, while functions stand alone.
- A message is to a method what a function call is to a function. Sending a message to an object calls the appropriate method (if the object has one—i.e., responds to the message).
- A selector partially identifies a method. When you send a message, the two most basic parts of it are the receiver, which is an object, and the selector. The method is found by looking it up, by the selector, inside the object.

Selectors are also values, just like numbers and pointers. The type of a selector is `SEL`. Every method has at least two arguments, the first two of which are hidden:

- The receiver of the message, as an argument of type `id` (pointer to an object) with the name `self`.
- The selector of the message, as an argument of type `SEL` with the name `_cmd`.
- Any other arguments the sender provided, which must match the arguments the method expects.

The two implicit arguments mean that, within a method, you can refer to the object that is responding to the message as `self` and, if necessary, you can refer to the selector of the message as `_cmd`. It's usually not necessary to use `_cmd`; C's own `__func__` (which is a constant provided and defined by the compiler, not a hidden argument) works better for most purposes. `self`, on the other hand, you will use all the time—specifically, any time you want your object to send a message to itself.

Since selectors are values, you'll sometimes see methods that take a selector as one of the explicit arguments. These methods will usually take an object in another argument, the idea being that you pass both an object to send a future message to, and a selector that identifies the message to send.

Another common place to see a selector as a value is as the value of a property. The "target-action" pattern in Cocoa and Cocoa Touch is probably the most common example: the action is one property in a pair, and its value is a selector. The target is the other member of the pair, and its value is an object. Once you've set both, the object with both of these properties—commonly a control, such as a button—will, in some future circumstance, send an action message with the appointed selector to the appointed object.

When using either of these, you'll usually pass a literal expression for the selector. A selector literal looks like this: `@selector(showFramistanList:)`. Note that despite the parenthetical marks, this is not a function call—it isn't valid to say "`@selector`" by itself, like you can with a function, and the full expression is a single literal value, just like `42` or `"Hello"`.

Selectors follow a simple format:

- One or more segments, with a colon after each.
- If there is only one segment, there may be no colon after it, in which case the selector must be a valid identifier (such as `foo` or `_foo` or `foo_123`), and a method that this selector would match should expect no arguments (besides the implicit `self` and `_cmd`).
- Otherwise, each colon (each segment) corresponds to an explicit argument. Each segment may have an identifier name, which is generally a good idea, but technically optional.
- The order of segments matters. `foo:bar:baz:`, `foo:baz:bar:`, and `bar:baz:foo:` are different selectors with no relation to each other. (Compare Python, where a method `foo`—solely identified by that much—may take arguments `bar` and `baz` in either order.)

Thus, these are all valid selectors:

- `foo` (takes no arguments)

PDFpen: Feature #5 – Sign Documents

The affordable PDF toolkit

Stop treating your PDFs like it's 1999.

What good is a 21st-century office if you have to print on paper to sign a document? Using PDFpen, you can add signatures or handwritten notations to any PDF.

It's great for forms and contracts. When you're finished, you can just email them back. Say hello to the modern, paperless office... and goodbye to your fax machine!

To see all the top 10 features of PDFpen, visit us at: www.smilesoftware.com/mactech

Smile

Software that's just right

- `foo:` (takes one argument)
- `foo:bar:` (takes two arguments)

And these are invalid:

- `foo:bar` (when there are multiple segments, every one must have a colon after it)
- `I am the very model of a modern Major-General` (not a valid identifier)
- (empty)

As with identifiers in C, case matters. `foo` is a different selector from `Foo`. The usual convention is that each segment has a lowercase initial letter, unless it's part of a set of initials (such as "URL"), with every word's initial letter and every set of initials capitalized. Examples from Apple's Foundation framework include:

- `length`
- `writeToFile:atomically:`
- `initFileURLWithPath:`
- `URLWithString:`

A selector can refer to an instance method, a class method, or both. There is no way to tell from a selector alone whether it should be addressed to an instance or a class (or whether it even matters—there are a few messages that all objects respond to).

Wrapping up

Now you should have a handle on object-oriented programming in Objective-C. To summarize:

- An "object" is an instance of a class (or, for a few purposes, a class itself—but usually just instances).
- A class has both an interface and an implementation, which exist in a header file and a module file, respectively.
- Objects talk to each other in messages.
- Every message includes the receiver of the message (an object) and a selector, along with the explicit arguments (if any).
- Any code, including functions, can send a message to an object, and you can send messages to any kind of object, including classes.
- An object's class determines which messages it responds to (i.e., what methods it implements).
- An object's class also determines what properties and instance variables it has—i.e., what other objects and values it owns and knows about.

Next month, we'll continue this look into Objective-C by digging into the syntax.

About The Author

Peter Hosey is a programmer of numerous open-source Mac applications. He answers programming questions on Stack Overflow and can be reached via Twitter as @boredzo.

by Boisy G. Pitre

The Sandman Cometh

How Apple's sandboxing security feature will affect your Mac apps now, and to come.

Introduction

We all live in a time when security is of paramount concern. Whether it's our vehicles, our homes, or our places of business, we are conditioned to think about things in terms of safety and security. This is no less true of our use of computers and mobile devices, which seem to be a fertile playground for all sorts of hackery and nasty business these days. When it comes to the major platforms, Apple has sported a veneer of impenetrability; viruses, Trojan horses, and other malware only happen on "that other platform." Although this perception has been tarnished recently with reports of surreptitious software targeting the Mac specifically, for the most part it still holds true.

Yet just because this has held true up to now doesn't mean that it will continue to do so. Even with the Mac currently having the advantage in the security department, it still makes sense for Apple to try to stay ahead of the curve. Security threats are always coming up and ever changing, and addressing them head on can keep our favorite platform ahead and winning the war. Such is the case with App Sandbox, Apple's technology that we will be discussing in this month's *Developer to Developer*.

It's A Scary World Out There

When you think about security on your Mac, it is important to consider how attacks against a computing system can occur. All applications run as processes on Mac OS X. In turn, each process runs with specific privileges based upon the type of user (admin or otherwise), and these privileges are granted and enforced by the operating system. Given that the kernel itself is the traffic cop for the computer it is managing, it makes sense that it should be the point at which operations are either allowed or denied.

In its simplest form, an application can simply start up, do some computation, and then exit. However, it is typically of little value to simply do something without interacting with the user in some way, either through showing visual information on the screen, or accessing a file, or printing a document. These interactions with the screen, or file system, or external devices make our applications richer and more useful. They can also make our applications more susceptible to attacks, allowing them to be used as agents for more sinister purposes. Recognizing this area of vulnerability and bringing in policies that can be enforced is the essence of sandboxing.

Life's A Lot Like A Sandbox...

Apple's use of a sandbox as a metaphor for application security makes a lot of sense when you think about how a sandbox is constructed. It's a rectangular shaped structure delineating a boundary, intended to keep material, in this case sand, within the bounds of the specific area. A "clean" sandbox is one that has only sand in the perimeter, and no sand outside the perimeter. Now, think of your application in such terms, and you can understand the intent of sandboxing: it strives to keep your app in best behavior with the rest of the system by giving it a well-defined set of boundaries in which it can work and play.

If you've been writing programs for a long time, life in the sandbox world can be a little limiting. Most of us take it for granted that we can access any file on the file system (assuming permissions allow us to), use devices such as cameras and microphones, open network connections over the Internet to other computers, and even host our own servers. How can we live without `fopen()` or `read()` to do I/O in C? That level of freedom is restricted in a sandboxed environment, and it can feel a lot like the walls are closing in on your application.

Those who have been doing iOS development are acutely aware of the restrictions placed on their applications. Mac developers on the other hand, must become accustomed to this brave new world, specifically if they want to reside in Apple's Mac App Store. That's because starting in November, Apple has announced that apps in the store will be sandboxed. That has a lot of developers concerned about how their apps will behave in such an environment, and rightly so. If an application does not properly plan for sandboxing, it may find itself suddenly crippled because services that once worked are now denied to it, and the user experience suffers.

Entitlements To The Rescue

With sandboxing, Apple's ultimate intent is to deliver security at the application level. Obviously, Apple recognizes how important it is for our applications to continue do the things that they always did to make them useful. Just because an application


The idea:
Turn surfboards into art
Serena Mitnik-Miller and Mason St. Peter of custom surfboard collective Two Birds Fly are living proof that when you're passionate about an idea, and you free it, beautiful things happen. Whether they're managing orders or creating promotional materials, working in a trusted productivity suite helps them take care of business and make sure their big idea takes flight.
Free the ideas.
Office:mac
HOME & STUDENT
Office:mac
HOME & BUSINESS
Introducing Microsoft® Office:mac 2011
Transcend platforms and free your ideas with Microsoft® Office for Mac 2011.
Learn more at officeformac.com/freetheideas

runs under a sandboxed environment doesn't mean that it should be restricted from accessing other files on the file system, or not use the camera or microphone, or not open a network connection. Our apps still need the freedom to do these things.

That freedom comes in the form of *entitlements*. An entitlement is a grant, or allowance of a specific feature to an application. Essentially, the application "asks" for permission to open a network connection, or use a USB device or gain access to photos or music in a user's home directory. Apple performs granting of these entitlements, and they will enforce this via the review process after your app is uploaded to the Mac App Store.

Let's say your app needs to access the Internet to obtain airport scheduling information on an airline's web service over the Internet. For your application to gain access to that server, it must declare use of the entitlement `com.apple.security.network` to do so. Likewise, if your app also needs to access the user's address book in order to book a flight. This too, requires an entitlement, specifically named `com.apple.security.personal-information.addressbook`. As you can see, the naming convention for entitlements uses the reverse domain name notation, with the common prefix being `com.apple.security`.

There are a number of entitlements that your app can apply for, including access to external devices such as the microphone (`com.apple.security.microphone`) and the camera (`com.apple.security.camera`). It's easy to see why this is the case: a program with the intent of doing bad things could use a camera or microphone device to snoop or eavesdrop on users, capture data, then either save it to a hidden file on the filesystem or send it off to a server somewhere on the Internet.

While Apple provides a considerable number of entitlements, there are certain operations that they have deemed candidates for *temporary* entitlements. A temporary entitlement is just that: it grants applications the ability to perform certain operations for an unspecified amount of time. Such temporary entitlements allow for sending of Apple Events, reading and writing in the home directory or even using absolute paths, and the ability to perform global Mach service lookups. These are considered risky operations that Apple will apparently allow on a case by case basis, since they reside outside the bounds of the normal sandbox. Since the word temporary is present, it is expected that these entitlements could be refined or even completely disallowed in some future release of Mac OS X.

So given all this discussion on entitlements, where are they stored? Entitlements reside in a file named appropriately, *Entitlements.plist* which is created for you in Xcode 4. To do this, simply select your target in Xcode 4 and then click the Summary tab. From there, select the entitlements you wish to use in your application from within Xcode 4, and check to enable entitlements and App Sandboxing (see Figure 1). These actions set the `com.apple.security.app-sandbox` key.

SHATTERED MEMORIES?

WE CAN HELP

Recover Your Lost Files Now!

We Provide Data Recovery Software And Services For All Types Of Media

LC TECHNOLOGY INTERNATIONAL

www.LC-Tech.com/mt

A GLOBAL LEADER IN DATA RECOVERY

Mac Universal

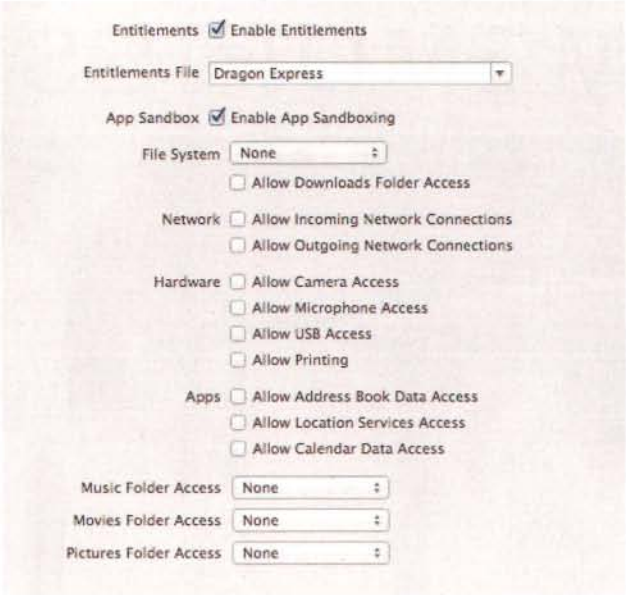


Figure 1. Entitlements pane in Xcode 4

The Mac App Store review process will match up your entitlements against your application's behavior to determine whether or not your application makes it into the store, or is rejected. And take note: asking for entitlements that your app doesn't need may get your app disapproved, so be judicious in the entitlements that you select. Make certain they are needed!

Files, Processes And The App Sandbox

For most of us, accessing files on the local file system will be the number one sandboxing issue that we will have to deal with. Depending on how your application works, it may ask the user to open a file, or save a file, via the ubiquitous file dialog box. For sand boxing, Apple has introduced a process based file dialog called *Powerbox*. This is a separate process that is spawned when the user wants to open or save a file. The process manages the interaction with the file system through the open/save dialog, and once a file is selected, your application is granted access to that file. How you can access that file depends on your entitlement. There are two entitlements: one for reading files (`com.apple.security.documents.user-selected.read-only`), and one for writing files (`com.apple.security.documents.user-selected.read-write`).

If your application does not need to write documents, it should only request the read-only entitlement. There are also folder specific entitlements available for applications to access user folders including documents, movies, pictures, and music.

Another interesting restriction is the application's notion of its viewable file system. Sandboxed applications will have full access to a folder and its contents located in the user's `Library/Containers` folder, but any file access outside of that will require entitlements.

If your application forks processes, be aware: application sandboxing is enforced by the kernel, an application's entitlements are inherited by any child process that it spawns through the fork/exec functions. This does NOT, however, apply to applications spawned by Launch Services using `LSOpen()` or other methods or functions. Applications launched in this manner will attain their own entitlements as specified by their entitlements list.

Viewing Entitlements For Existing Apps

If you have Lion, you can view the list of sandboxed apps currently running on your system by launching Activity Monitor and ensuring that the "Sandbox" column appears in the table, as shown in Figure 2. You can also check for sandbox violations by checking the Console application for any logging information from the sandboxd daemon.

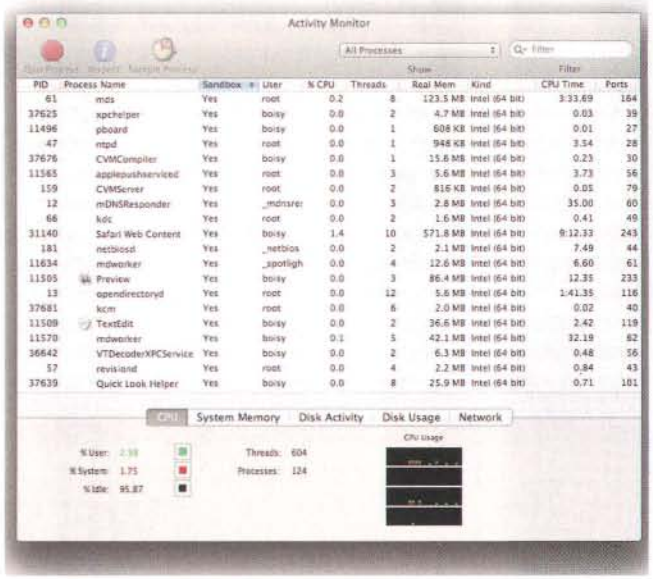


Figure 2. Activity Monitor showing Sandboxed Applications

It is also helpful to see just what entitlements are being used on existing sandboxed apps. Use the `codesign` command line utility to inspect the entitlements for applications that are sandboxed, like so:

```
$ codesign -d -v --entitlements - /Applications/TextEdit.app/
Executable=/Applications/TextEdit.app/Contents/MacOS/TextEdit
Identifier=com.apple.TextEdit
Format=bundle with Mach-O universal (i386 x86_64)
CodeDirectory v=20100 size=987 flags=0x0(none) hashes=41+5
location=embedded
Signature size=4064
Info.plist entries=30
Sealed Resources rules=11 files=10
Internal requirements count=1 size=344
??qq?<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
```

```
<plist version="1.0">
<dict>
    <key>com.apple.security.app-sandbox</key>
    <true/>
    <key>com.apple.security.files.user-selected.read-
write</key>
    <true/>
    <key>com.apple.security.print</key>
    <true/>
    <key>com.apple.security.app-protection</key>
    <true/>
    <key>com.apple.security.documents.user-selected.read-
write</key>
    <true/>
</dict>
</plist>
```

Here, the TextEdit application is shown with the `com.apple.security.app-sandbox` key set to true, along with the four entitlements that it needs in order to perform its duties. For a list of available entitlement keys, consult the *Code Signing and Application Sandboxing Guide*, available from Apple's documentation website.

Mitigating Risk With XPC Services

Another direction that Apple is encouraging apps to take in the name of security is XPC services. The idea behind XPC services is that certain functionality in our applications can be refactored into their own helper applications, doing one or perhaps more specific things on behalf of any application. It's similar to the object-oriented approach that we take in our applications, except the functionality for an XPC service resides in a separate process.

There is also a security aspect to using XPC services. Since sandboxing is a process-wide enforcement mechanism, entitlements apply to the entire process, not just parts of it. You cannot say that this section of my code can have entitlement A while another section can have entitlement B. It's all or nothing.

Breaking out functionality into separate processes allows a separate sandbox environment for your main application and all of its helper processes, so different entitlements can apply to different programs. An example that Apple has provided in their material is an application that downloads a file from the Internet and then compresses it. The application orchestrates the creation of the download task, which is a separate process that contains entitlements to make a network connection –nothing else. Using some form of interprocess communication (IPC), the download task then passes the acquired information to a compressor task whose job is to create a file with the compressed representation of the downloaded data. Hence, the download task only needs the entitlement to write to a file chosen by the Save dialog.

The reasoning is that by breaking this functionality into separate processes and compartmentalizing functionality, it makes the overall application more secure. Since each XPC service is a process, its own entitlements are enforced by the kernel. There's also another added benefit of XPC services: reuse of specific helper applications among a group of applications. This can be very useful if you have functionality that spans a number of applications that you support.

Something To Keep You Up At Night

While app sandboxing has a lot of advantages and gives us peace of mind, there are some downsides to implementing it, especially for us as developers. The obvious elephant in the room is whether or not Apple will eventually require all applications, even those not purchased from the Mac App Store, to run in a sandboxed environment. This remains to be seen. Obviously the ramifications of such a move would be significant and widespread. There are many apps that would require a good bit of work to behave well with sandbox restrictions. While this particular scenario hasn't been broached by Apple, it is something to consider as time goes on.

Summary

It's obvious that sandboxing will become a more important part of our work as application developers in both the near and longer term. Security is a moving target that doesn't stay still, so we can expect more and more changes with regard to entitlement availability and granularity. It's going to be interesting to see how it all unfolds.

I would encourage you to examine your apps and identify behaviors (file & network access, Apple Event use, etc.) that may need to be addressed through the use of entitlements, even if your app doesn't currently sell in the Mac App Store. If you have an Apple Developer Account, I would also recommend viewing the videos from WWDC 2011 entitled "App Sandbox and the Mac App Store" and "Introducing XPC."

Bibliography and References

Apple, *Creating XPC Services*, http://developer.apple.com/library/mac/#documentation/MacOSX/Conceptual/BPSystemStartup/Chapters/CreatingXPCServices.html

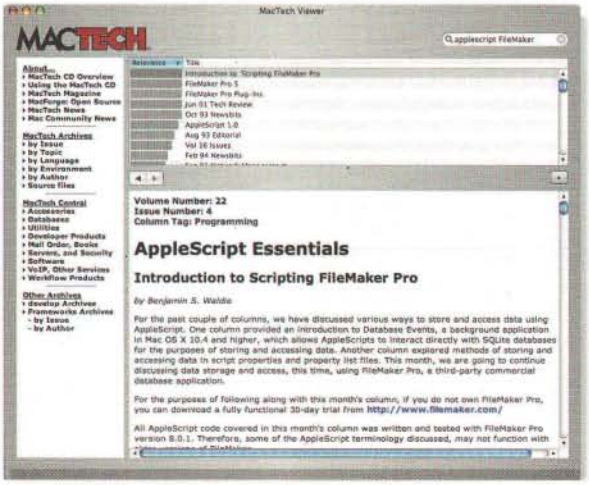
Apple, *Code Signing And Application Sandboxing Guide*, http://developer.apple.com/library/mac/documentation/Security/Conceptual/CodeSigningGuide/CodeSigningGuide.pdf

About The Author

Boisy G. Pitre is a senior software engineer with Nuance Communications, where he works on cutting edge speech recognition software for the Mac. He holds a Master of Science in Computer Science from the University of Louisiana at Lafayette, and resides in the quiet countryside of Prairie Ronde, Louisiana. Besides Mac programming, his hobbies and interests include retro-computing, ham radio, vending machine and arcade game restoration, and playing Cajun music. You can reach him at boisy@tee-boy.com.

The MacTech DVD - Volumes 1.01-27.03
is packed with more than ever before -- over 3300 articles from more than 300 issues **(1984 - March 2011)** written by almost 900 experts, all 29 issues of Apple's *develop*, 21 issues of *FrameWorks* magazine, all the source code, MacTech Viewer, working applications, full documentation, demos for techs, ***and more!*** The latest version includes all kinds of third-party applications, videos and demos.

See for yourself why *MacTech Magazine's* DVD is the best information source for Macintosh techs and developers. Search quickly through over 27 years of of great information provided by *MacTech*. Information to save you time, and make your life easier.



Requires Mac OS X v. 10.4.5 or later

Toll Free 877-MACTECH, Outside US/Canada: 805-494-9797 • http://www.mactech.com/dvd/

The Whys and Hows of Lion's Recovery HD

What is it, and why do we need it?

by Rich Trouton

One of the new features in 10.7 is the Lion Recovery feature. The general idea is that, if your boot volume runs into trouble, you can boot from the hidden **Recovery HD** partition on your hard drive, or NetBoot from Apple's Lion Internet Recovery (only available to Mac models introduced in July 2011 or later.) Once booted to the **Recovery HD** drive, you'll have access to all of the tools you need to run diagnostics, repair your disk, restore from a Time Machine backup, or even reinstall 10.7.

Since the average Mac admin has other tools available to boot and fix the Mac, there are some questions about it that may arise, like "Why is Recovery HD there?", "How do I create a Recovery HD partition?", "Can I run without it and still be supported?" and "How can I remove it?"

Why Recovery HD is there

The Recovery feature is there for a couple of reasons, one of which is obvious and another that is not as obvious. The obvious reason is that it provides a safety net that's always available for your Mac to use. Even if your OS is non-functional, you can boot from **Recovery HD** by holding down Command-R during startup. Once booted from **Recovery HD**, you can use its tools to fix your Mac or recover your data from a boot drive with problems to another drive attached to your Mac. More important, as long as your Mac has a connection to the Internet (and thus to Apple's servers), you can reinstall an OS that will boot your Mac without having to hunt for install media.

The non-obvious reason for **Recovery HD** has to do with FileVault 2. FileVault 2 encrypts your boot partition, but your Mac still needs an unencrypted space to boot to and allow access to the encryption unlock tools. The **Recovery HD** partition serves as the needed unencrypted space. The FileVault encryption process will check before beginning the encryption to see if the **Recovery HD** partition is there and will not start the encryption process if it's not there.

Creating your own Recovery HD partition

One of the more mysterious processes in Lion is how the **Recovery HD** partition is created in the first place. There's no "Create Recovery HD" button in 10.7's Disk Utility and there's no obvious command available to create it after the fact. This is especially important when imaging, as a number of imaging tools (Apple's NetRestore among them) don't create the **Recovery HD** partition when laying down an image. Fortunately, there's a way to create a **Recovery HD** partition without having to install Lion entirely.

Prerequisites:

Install Mac OS X Lion.app from the Mac App Store

A separate 3GB or larger drive on which to create the **Recovery HD** partition

1. Wipe the drive and use Disk Utility to repartition with one GUID partition. In my case, I named the drive **RecoveryBuild**.

2. Mount the **InstallESD.dmg** file from the Install Mac OS X Lion.app

3. Open Terminal and dump the installation choices for a standard OS install to an xml file by running the following command:

```
installer -verbose -pkg /Volumes/Mac\ OS\ X\ Install\
ESD/Packages/OSInstall.mpkg -tgt /Volumes/RecoveryBuild -
showChoiceChangesXML > /tmp/choices.xml
```

4. In whatever editing program you prefer, edit **/tmp/choices.xml** so that all of the **attributeSetting** keys are set to zero, except for the **EssentialSystemSoftware** and **EssentialSystemSoftwareGroup** dicts, in which case the **attributeSetting** key is left at 1. This is to speed the process up a bit by not installing all of the packages.

5. In Terminal, run the installer command with the modified choices.xml file:

```
sudo installer -verbose -pkg /Volumes/Mac\ OS\ X\ Install\
ESD/Packages/OSInstall.mpkg -tgt /Volumes/RecoveryBuild -
applyChoiceChangesXML /tmp/choices.xml
```

What this process does is to install a fully functioning **Recovery HD** partition and leave a non-functional Lion installation on the main partition. Once you have the **Recovery HD** partition created, you can deliver an image with NetRestore or another imaging tool to the main partition.

Running Lion without a Recovery HD Partition

It is possible to install Lion without having an accompanying **Recovery HD** partition created. Two Apple-identified scenarios

Parallels

My Mac does do Windows.

Parallels Desktop® 6 for Mac.
Simply Faster, Smarter and More Powerful Than Ever!

- **New!** Get full control of your virtual machine with our all-new Parallels Mobile app for iPhone/iPad
- **New!** Take advantage of all of the capabilities of your 64-bit Mac and enjoy our fastest virtual machine performance yet
- **Enhanced!** Experience brilliant graphics capabilities in Windows applications whether you're a gamer, architect, designer or engineer
- **New!** Immerse yourself in your favorite games, music and videos with rich 5.1 Surround Sound

For business or for home, Parallels is the #1 choice of customers worldwide*

To learn more, visit www.parallels.com/desktop today.

Award winning reviews!

*No. 1 claim based on NPD Techworld Reports for the period 8/1/06 - 09/30/10

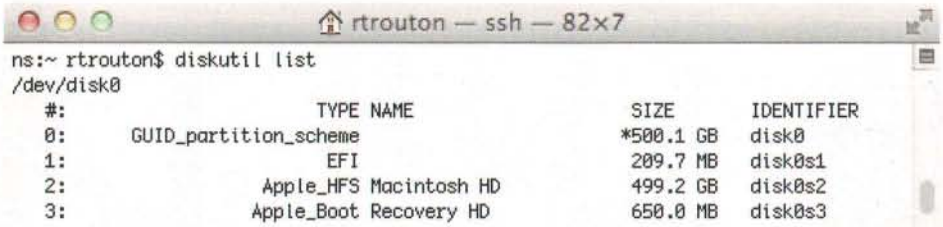


Figure 1 – diskutil listing showing recovery partition.

where the **Recovery HD** partition may not be created are the following:

- The disk you are installing Lion on is a RAID volume
- The disk has a non-standard Boot Camp partition setup, where further partitioning was performed after running Boot Camp Assistant, or the configuration that Boot Camp Assistant created was manually modified.

Apple will support a Mac that's running Lion without a **Recovery HD** partition, but it would not be possible to encrypt that drive with FileVault 2. In the event that you would need the utilities that **Recovery HD** provides, Apple recommends having another drive with a **Recovery HD** partition available.

Removing Recovery HD

In the event that you decide you don't need to have a **Recovery HD** partition on your Mac(s), it can be removed by running a few commands in Terminal.

1. To get the identifier of the **Recovery HD** disk partition, run the following command:

```
diskutil list
```

You should see output similar to Figure 1:

In this case, the identifier is **disk0s3**.

2. Next, use the identifier to erase the **Recovery HD** partition and rename it as a drive named **RemoveMe**:

```
diskutil eraseVolume HFS+ RemoveMe disk0s3
```

3. If desired, you can run the following command to reclaim the space by merging your main and recovery partitions together.

```
diskutil mergePartitions HFS+ "Macintosh HD" disk0s2 disk0s3
```

Note: When merging two or more partitions, always make sure to have a current backup of your data.

Conclusion

With any new OS comes changes, and Lion has produced a number of them for Mac admins to work on and work with. The Recovery HD partition is one that your users will hardly see, but may prove to be a significant factor in your deployment strategies for Lion. Hopefully, the information provided should help you decide whether or not to include Recovery HD when you roll out Lion in your own environment.

MU

About The Author

Rich Trouton has been doing Macintosh system and server administration for over a decade and has supported Macs in a number of different environments, including university, government, medical research and advertising. His current position is providing support for Howard Hughes Medical Institute's Janelia Farm Research Campus in Ashburn, Virginia.

codefortytwo
software

Back up everyone, everywhere.

CRASHPLAN™
Continuous Backup for Everyone

The easy, automatic way to protect all your data.

Individuals, businesses and enterprises around the world count on CrashPlan+, CrashPlan PRO and CrashPlan PRO*e* to keep their data safe.

You can too! **Try it for free for 30 days.**

Peace of mind is just a few clicks away at crashplan.com

Code 42 Software • one main st. se #400, minneapolis, mn 55414 • 612.333.4242

KosmicTask

Sharing script-based functionality across the network

by Jonathan Mitchell, Mugginsoft LLP

Introduction

KosmicTask is an integrated scripting environment for OS X that makes it easy to develop, test and share script based tasks across a network. At its most abstract, a task is simply a named operation performed at the request of a user. At its most particular, it is a carefully authored sequence of instructions in a designated scripting language. The KosmicTask environment provides facilities that help users to locate and execute the functionality they need and allows authors to create and update tasks in response to user requirements.

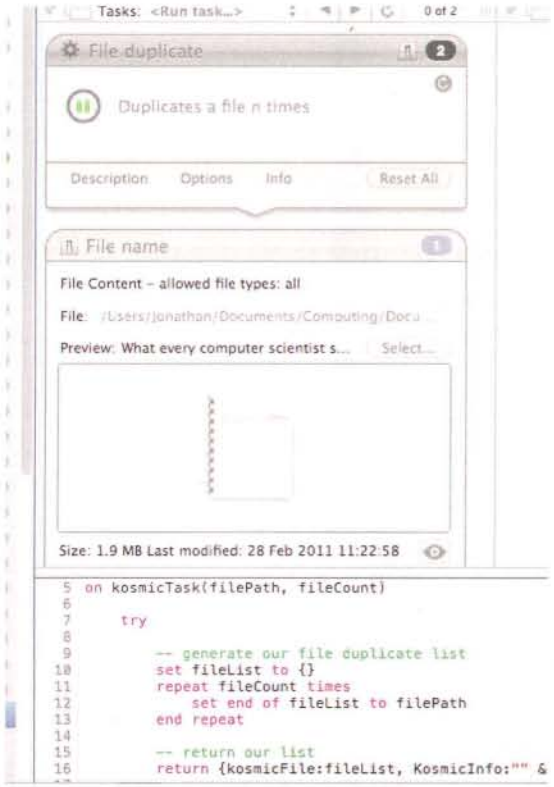


Figure 1. KosmicTask executing a file duplication task on a remote machine.

A task may have a number of defined inputs of specific types (text, number, file etc) and may return a result to the user. KosmicTask supports simple text based results but also features extensive support for complex data results that can be saved in various common formats. Complex results can also be used to return entire files, which means that highly effective file processing tasks can be written in any of the supported scripting languages.

A default installation of OS X provides access to a wide range of scripting languages and KosmicTask provides plug-in support for all of them (the shells, AppleScript, Java, JavaScript, Perl, PHP, Python, Ruby and Tcl). In addition, the application also includes a number of embedded language plug-ins that greatly extend scripting possibilities (C, C++, Lua, F-Script, JSCocoa and LuaCocoa). It is hopefully obvious that KosmicTask takes a pretty liberal view of what constitutes a viable scripting language and Kosmic logic says that it has very little to do with the language itself and common usage patterns and everything to do with accessibility. Thus, while the likes of C++ will never rival Bash or Perl as a general system administration language, KosmicTask provides traditional scripting ease of use to a robust interpreted C++ implementation that makes experimenting with and learning about C++ enjoyable and instructive.

In addition to providing additional language plug-ins, the application also provides a means of extending what can be achieved with scripting through the use of several Cocoa bridges. The Cocoa framework provides the broad underpinnings of the majority of OS X applications and provides a wealth of functionality that can be called upon to build genuinely powerful tasks. An additional benefit provided by the Cocoa bridges is that they provide access to an alternative means of automating applications. OS X includes long standing support for application automation via AppleScript which, although highly capable, is undoubtedly idiosyncratic. The Scripting Bridge is an OS X technology that permits scripting languages other than AppleScript to engage in application automation.

Experience the Power of Control in 3D

NVIDIA 3D VISION READY

Mac

IOGEAR 4-Port Dual-Link DVI KVMP Pro Switch

IOGEAR is pushing the envelope again with this powerful KVMP supporting high-end 3D graphics with NVIDIA® 3D VISION. This KVMP is compatible with most gaming keyboards and is equipped with an integrated 2-port USB 2.0 hub for peripheral sharing. Now, users can control up to four computers and share multiple USB peripherals from a single keyboard, monitor, and mouse console. This powerful KVMP can increase efficiency and save equipment cost for applications where high resolution, large or widescreen monitors are required such as:

- CAD (Computer Aided Design)
- CAM (Computer Aided Manufacturing
- Medical / Surgical / Diagnostic Displays
- Desktop Publishing / Graphic Design
- Video Editing
- High-End Gaming

GCS1204G

Share a professional quality monitor, keyboard, and mouse and 2.0 or 2.1 audio support between four USB / DVI computers

Supports high-end 3D graphics with maximum video resolutions of:
- DVI Dual-Link: 2560 x 1600
- DVI Single-Link: 1920 x 1200

DynaSync* technology reads and remembers the monitor's parameters to eliminate delay or change of video resolution when switching between, or booting computers

Go Online!

For more information, go to http://go.iogear.com/ms/GCS1204G

*IOGEAR's DynaSync provides the EDID support for Windows 7 OS

IOGEAR®

Convergence through Connectivity

Target Audience

The concept of a computational task is universal and the intended target audience is correspondingly broad. System administrators will find that the application offers a secure means of providing script based maintenance and tasks can be created, edited and executed remotely. The file transfer and automation support features will appeal to commercial and creative users who wish to share or centralize file processing operations. Software professionals can build libraries of executable code samples and quickly prototype design ideas. The application also makes a great platform for implementing the sort of remote data acquisition projects commonly encountered in science and technology.

Non-technical end users will obviously benefit from accessing tasks targeted to their needs and team mangers can easily review the available tasks to ensure that common processing and workflow functionality remains relevant and up to date.

The application is also a great place in which to learn about scripting and experiment with different approaches and solutions. The Internet provides access to vast amounts of downloadable script code in a plethora of source languages. KosmicTask makes it practical for anyone, from the determined home user to the gnarled sysadmin, to take relevant source material in a perhaps unfamiliar scripting language and customize it to their needs.

Task User Overview

From a user's point of view a task is a resource that exists somewhere on the network and somehow or other does what needs to be done. KosmicTask features a familiar iTunes like multi-tabbed user interface with the following features:

Network task browser. Users can browse connected machines for suitable tasks or use the Spotlight powered search facility to search across all tasks available on the network. Tasks can be displayed according to group membership or alphabetically.

Automatic network discovery. Bonjour is used to ensure that users have access to available task resources as machines connect and disconnect from the network.

Secure task sharing. Tasks can be made accessible to all network users or only to those who can authenticate. Network data privacy is accomplished using TLS.

Support for multi-tasking. Multiple tasks can be run simultaneously in individual tabs or tasks can be detached into individual windows.

Comprehensive history retention. Each tab retains a detailed history of all the tasks executed within it and all results generated. An application wide history facility retains a long-term record of task activity that can be used to recall previously executed tasks and their inputs.

Figure 2. Simultaneously calculating PI to 10,000 places on 3 different machines.

Tasks are executed on the machines that host them. This is often an essential requirement when dealing with application automation as the target application has to be locally accessible. Tasks may generate results, such as text, imagery or files, all of which can be browsed and saved to disk or forwarded to other applications. For a comprehensive explanation of all aspects of the user interface see the online help book at www.mugginsoft.com/kosmictask/help.

Task Author Overview

From a task author's point of view a task is a piece of functionality that makes use of particular scripting technologies. Most IT and computer professionals will have a preferred set of skills, tools and resources to call upon whilst non professional task authors will most likely start with some scavenged example code. KosmicTask includes a range of author features that will appeal to experienced and novice authors alike:

Sample tasks and templates. The application includes a number of fully functional sample tasks in a range of scripting languages. A powerful template facility provides a range of short templates for each of the supported scripting languages that demonstrate how to implement key pieces of functionality (how to access arguments, how to return a file, how to automate an app, etc). These templates serve as starting points for new tasks and as essential reference material.

Intuitive task editing and testing. The task editor provides a simple environment in which to define tasks, edit code and perform test executions. The code editor features language aware syntax coloring and, if appropriate, a build stage may be invoked to check for syntax errors or perform compilation.

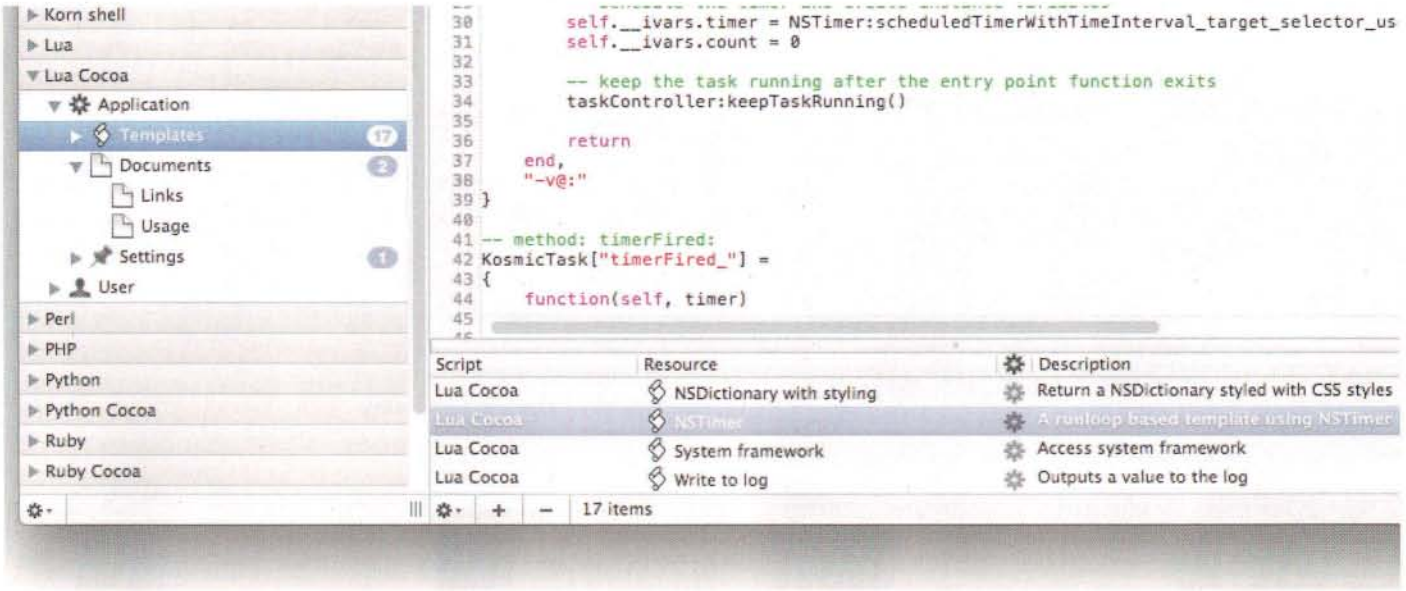


Figure 3. Application resource browser displaying the list of LuaCocoa templates.

Zero configuration building. Most tasks can be created without any configuration, especially if they are based on a template. Any build or runtime configuration that is required can be applied at the level of an individual task or as a language default.

Remote access. Authors who can authenticate as a valid OS X user on a network instance of KosmicTask can create and edit tasks remotely. This provides an efficient means for configuring tasks on shared application servers and for providing remote task support.

Integrated resource browser. The resource browser provides access to application supplied templates, documentation and language configuration settings. In addition, task authors can define their own templates and create additional documentation.

Hello Kosmos!

That's it for the formal introductions! Now it's time get personal and saying hello is a typically terrestrial way to begin. First, if you don't already have KosmicTask, you should download the latest version from http://www.mugginsoft.com/kosmictask/download. To create a new task from within KosmicTask select the Admin button from the Task View control (if you have selected a shared instance of KosmicTask then you will be asked to authenticate). In the Task section of the edit window we assign a name, group and various other properties to our task. We may also optionally define a number of task inputs. Selecting the Edit button will display the code editor and because our task is currently empty the template sheet will be displayed. We navigate to the Ruby application templates and select the Hello kosmos item as follows:

Listing 1: Doing the introductions in Ruby

```
# send result to stdout
puts "Hello, kosmos!"
```

Every supported scripting language features a similar Hello kosmos template that simply returns a string of characters to the user when the task is executed. In the case of Ruby and other traditional command line scripting languages, this is achieved simply by printing or explicitly sending data to stdout. In addition to generating results, many tasks will require input. KosmicTask provides a number of standard input types any number and combination of which can be defined as the input set for a particular task. The following example simply echoes the first task argument back to the user:

Listing 2: Echo a task input in Ruby

```
# echo first command line argument to stdout
puts "#{ARGV[0]}"
```

If we define a textual or numeric input for our task then the above template will simply echo that exact input back to the user. However, if we define a file input and select a text file, say `test.txt`, then the result of executing the above template will be something like:

```
/Users/Jonathan/Library/Caches/com.mugginsoft.kosmictaskserver.
files/CVxLZ5/test.txt
```

This reveals that the contents of our input file have been copied into a temporary cache (note that the original file name is preserved). This behavior enables input files sent from both local and remote machines to be accessed locally by our task script. It also has the additional benefit of ensuring that tasks operate on copies of input files and not on originals. Echoing the actual content of our input file is a relatively simple matter:

Listing 3: Echo a task input file in Ruby

```
# echo first command line argument to stdout
path = ARGV[0]
puts "input file path = #{path}"
```

Is your subscription solution provider as flexible as you are?

Complete control of sales process • Upgrade/downgrade capabilities • Trial period support
Custom messaging and terms • Customer-controlled account management
No third-party service requirements • Real-time reports and notification

esellerate®
Digital River™

Get more out of your Subscription Solutions.
Less hassle. More profit.

www.esellerate.net/mactech

+1 402-323-6600 x39701

```
open(path) |f|
    # echo content even if the file is empty.
    puts "input file content = #{f.read}"
    }
```

Some tasks may only require input files, but to compose a fully functioning file processing task we will need to return a file to our user. In order to do this our task will have to return a complex result.

Complex Result Handling

Tasks can generate two types of results:

Simple Result. A simple result is a string of characters representing either a block of text or a textual representation of a number or other object such as a date. All of the supported scripting languages can return simple results.

Complex Result. A complex result is a structured object that contains one or more parts and can retain type information for each part. Complex results may also incorporate styling information that can be used to format the displayed result. All of the supported scripting languages can generate complex results.

In the examples above, our Ruby powered tasks returned simple textual results by printing to stdout. In order to generate a complex result, we format our result using YAML, a flexible, human-readable data serialization format. The next example returns a list (aka an array) of the solar terrestrial planets:

Listing 4: A complex block YAML result in Ruby

```
# return a YAML block format list
puts "---"
puts "- Mercury"
puts "- Venus"
puts "- Earth"
puts "- Mars"
```

YAML also defines an optional inline formatting style (which is equivalent to JSON):

Listing 5: A complex inline YAML result in Ruby

```
# return a YAML inline format list
puts "---"
puts "[Mercury, Venus, Earth, Mars]"
```

Complex results can be sorted and/or exported in various data formats. As well as defining a list, a complex result can return a dictionary (aka a hash, map or associative array) of keyed items and it is this facility that we will use to send the content of a file as part of our task result. The following example takes a screenshot and returns the captured image file as part of the result:

Listing 6: Returning a result file in Ruby

```
# file created in task current directory will be automatically
deleted
file = "capture.png"

# capture screen shot to file
```

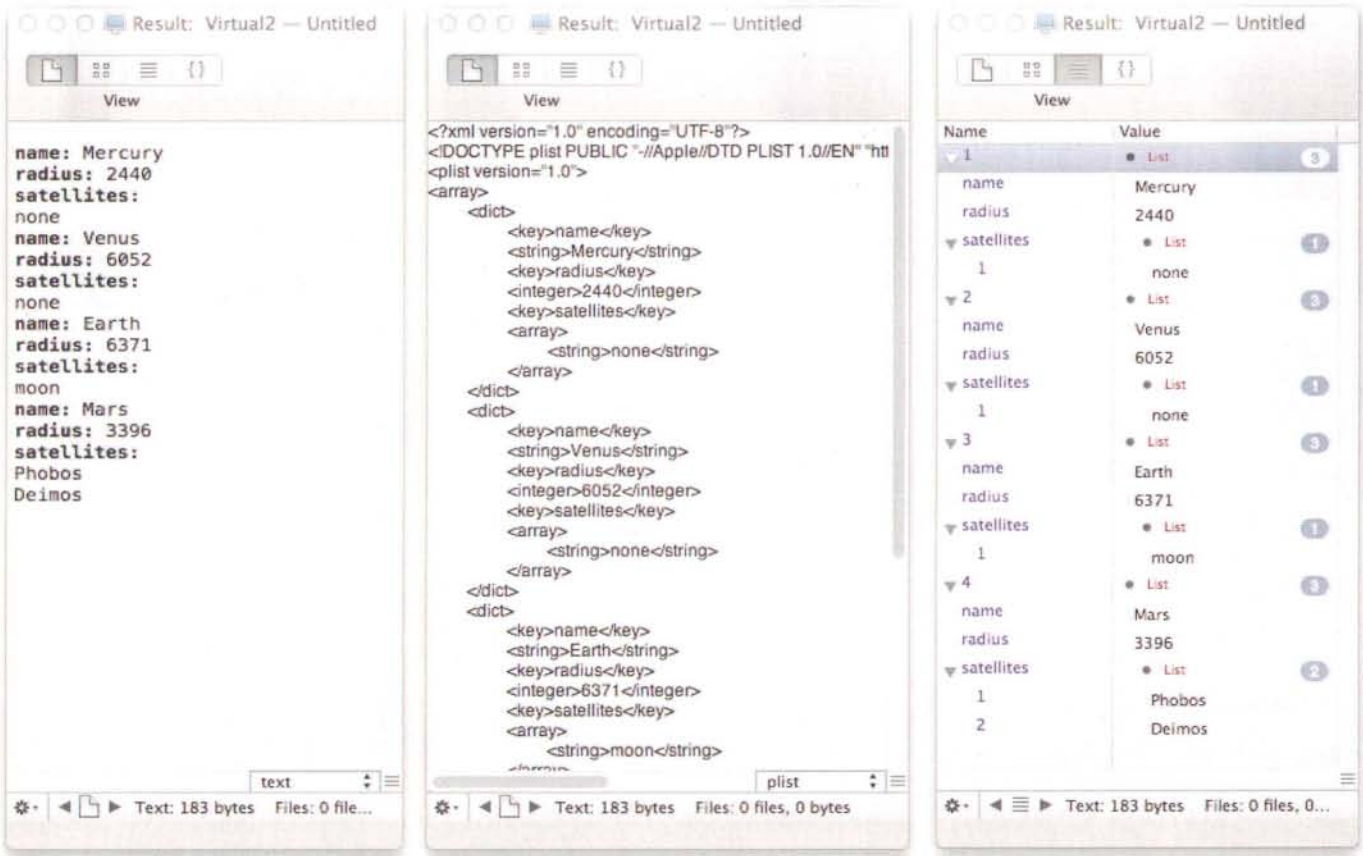
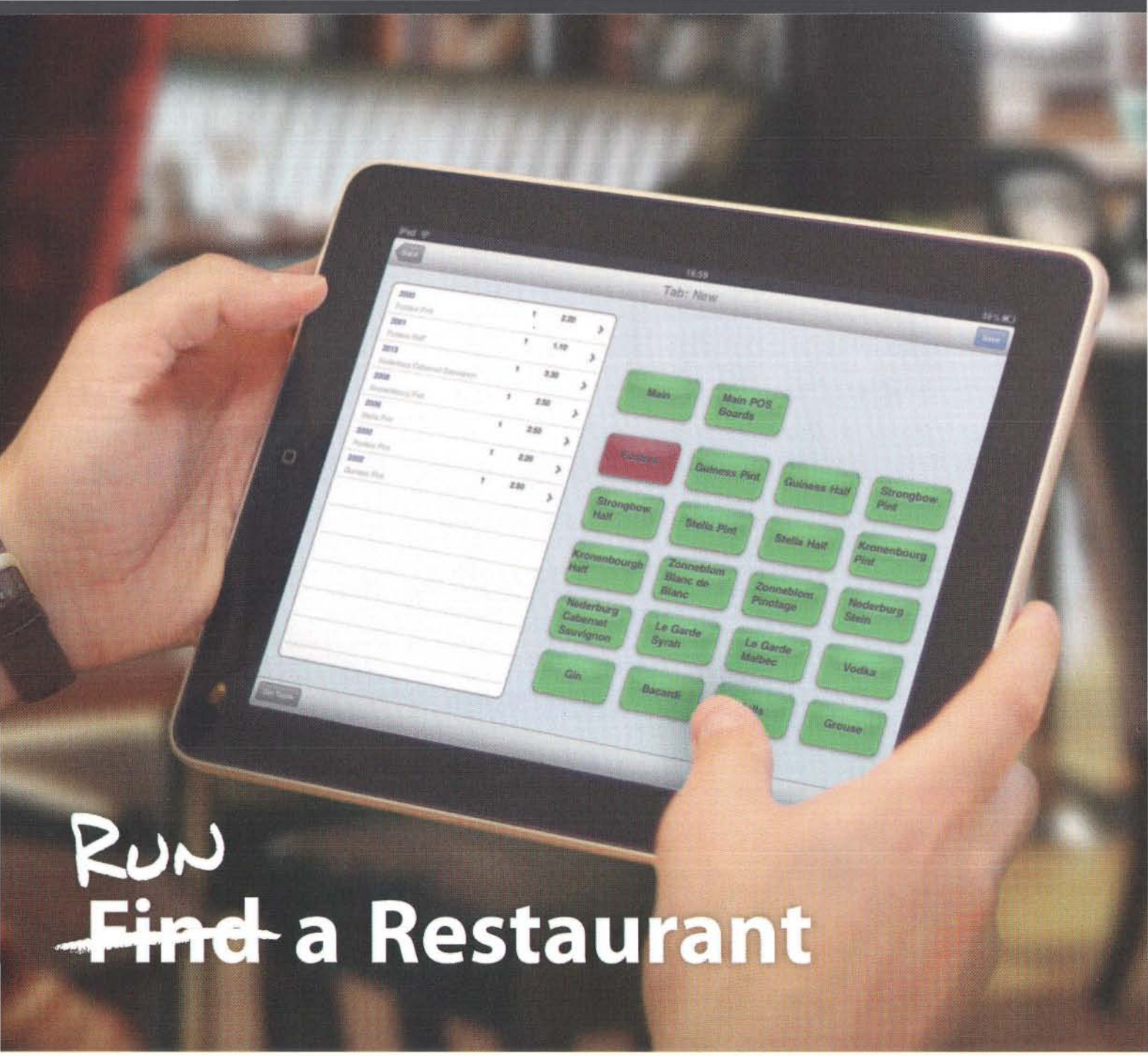


Figure 4. A complex result displayed as text, as a plist and as an outline.



RUN ~~Find~~ a Restaurant

Ordering | Kitchen Printing | Reporting

Tabs opened from graphical table view. Fast, accurate orders, including special requirements. Wireless printing to kitchen and for final bill. Improve your customer service with Restaurant by HansaWorld, for iPad.

Books by HansaWorld

Enterprise by HansaWorld

Integrated CRM | Real-time Mobile | Real-time Consolidation | Integrated Webshop | Built-in Document Management | Powerful Analysis

HansaWorld USA, Inc. 20900 NE 30th Avenue #414, Aventura, FL 33180, USA, Phone: +1 646-459-0603, Email: usa@hansaworld.com

```
system "screencapture -t png " + file

# return a YAML inline format dictionary with filename
puts "---"
puts "{kosmicFile: #{file}, kosmicInfo: file returned}"
```

The key point to note here is the use of the `kosmicFile` dictionary key. KosmicTask parses every task result and when it encounters the file key it appends the content of the file to the result response. The `kosmicInfo` key merely identifies a message to accompany the file. Now we are in a position to compose a task that accepts a text file as input and generates a processed file as a result:

Listing 7: File processing in Ruby

```
# get the input file path
path = ARGV[0]

# read the input file text
data = ""
open(path) {|f| data = f.read}

# write the reversed data to a file in the current directory
open("reverse.txt", 'w') {|f| f.write(data.reverse)}

# return our processed file
puts "--- {kosmicFile: reverse.txt, kosmicInfo: file content
reversed}"
```

The example above returns a single file. To return multiple files we simply return a list of filenames as opposed to a single filename. The following example simply returns 3 identical copies of the input file:

Listing 8: Returning multiple files in Ruby

```
# get the input file path
path = ARGV[0]

# return 3 duplicates of our input
puts "--- {kosmicFile: [#{path}, #{path}, #{path}] }"
```

It's nearly time that we left planet Ruby. Before we go we will consider one last point. The examples above demonstrate the use of YAML for complex result structuring within the context of Ruby, but it is important to note that the same principle applies to many of the other supported scripting languages including all the shells, C, C++, Java, JavaScript and more. In actual fact, explicit YAML is provided as a fallback position for languages such as the shell script languages that often do not support native structured data formats. Although directly outputting YAML is functional, it is hardly elegant and in many languages, including Ruby, it can be avoided altogether by the use of a KosmicTask controller object. This approach enables our task to return a native data structure as a result rather than having to generate YAML. A controller object is made available to every suitable task and works by automatically serializing the native data structure into the appropriate YAML, using the `printObject` method, before sending it to stdout. The example below generates a complex native Ruby result containing a list of dictionaries:

Listing 9: Returning a native complex result in Ruby

```
# access the controller
require "KosmicTaskController"

name = "name"
radius = "radius"
moons = "satellites"

# Mercury
satellites = Array["none"]
Mercury = Hash[name => "Mercury", radius => 2440, moons =>
satellites]

# Venus
satellites = Array["none"]
Venus = Hash[name => "Venus", radius => 6052, moons =>
satellites]

# Earth
satellites = Array["moon"]
Earth = Hash[name => "Earth", radius => 6371, moons =>
satellites]

# Mars
satellites = Array["Phobos", "Deimos"]
Mars = Hash[name => "Mars", radius => 3396, moons =>
satellites]

# assemble the solar terrestrial planets
Planets = []
Planets.push(Mercury)
Planets.push(Venus)
Planets.push(Earth)
Planets.push(Mars)

# print native object as YAML
KosmicTaskController.printObject(Planets)
```

Results can be opened in separate result windows and displayed in various formats: (Figure 4, below).

Farewell dear Ruby!

Now, I like Ruby and you probably like Ruby but, having established some basic Kosmic principles, it's time to move on. The KosmicTask language plug-in architecture supports two process models when it comes to executing tasks. Traditional command line based languages run out-of-process with the task runner that executes each task. Other scripting languages run in-process with the task runner and this affects them in two distinct ways:

Input handling. In-process tasks can send their inputs as named parameters to a specific function. This makes accessing and identifying inputs easy.

Result handling. In-process tasks can generally return native data structures as objects without having to resort to YAML or utilize a controller. In the case of the Cocoa bridges (PyObjC, RubyCocoa, JSCocoa, etc) it also means that tasks can return native Cocoa objects as results. That last bit is, in my book, a humdinger.

Now, I don't like AppleScript and you probably don't like AppleScript but the fact is that it is the primary automation language for OS X. AppleScript runs in-process so the points outlined above apply to the following example, a duplicate of our previous file processing task:

Listing 10: File processing in AppleScript

```
on KosmicTask(theFile)
```

```
    try
        -- read the input file text
        open for access theFile
        set fileContents to (read theFile)
        close access theFile

        -- get a result file object from KosmicTask.
        -- the file will be automatically deleted when the task
ends.
        tell application "KosmicTask"
            set resultFile to result file with name
"reverse.txt"
        end tell

        -- reverse file content and write to file
        set fileContents to reverse of characters of
fileContents as text
        open for access resultFile with write permission
        write fileContents to resultFile
        close access resultFile

        -- return our processed file
        return {kosmicFile:resultFile, kosmicInfo:"text
reversed"}

    on error errorMessage number errorNumber
        return {kosmicError:errorMessage}
    end try
end KosmicTask
```

So in this case, our task entry point is defined by the `KosmicTask` function and its single named input argument (the actual name of the entry point function can be configured in the task settings if required). Our file is processed as before and returned within a record, the AppleScript equivalent of a dictionary (it's a coincidence that the AppleScript record syntax looks a lot like inline YAML). We also use the `kosmicError` key to explicitly flag an error. And at this point it might be a good idea to think a bit more about file handling.

We have already mentioned that input files get copied to a cache and in our file processing examples we created new files and returned them as results. A natural concern would be that a busy task server would soon accumulate a lot of file detritus left over from task executions. However, there is nothing to worry about. Cached input files are deleted as soon as the task ends and as long as the task creates new temporary files in the recommended manner they will also be deleted when the task completes. This last behavior is process type dependent:

Out-of-process tasks. Prior to execution a temporary directory is created for the task and set as the current directory. Any files written to the current directory are considered transient. When the task ends the entire temporary directory is removed.

In-process-tasks. Tasks generally have access to a controller object that responds to the `resultFileWithName` method and returns a transient temporary file path. AppleScript, however, targets the application itself, as can be seen in the previous example.

Files received by a user as part of a result remain available to be permanently saved until the task tab or window referencing them closes, at which point they are deleted. AppleScript would not be a first choice for tasks involving complex file operations but it is good at automation, so here is an example that includes both. It accepts a Pages '09 text-only document as input and returns an equivalent HTML file (this is accomplished by saving the Pages document as Rich Text Format first and then using the `textutil` command line utility to convert the RTF to HTML):

Listing 11: Automation in AppleScript

```
on KosmicTask(pagesDocFilePath)

    -- we need a path to save our RTF document file into.
    -- the easy way is simply to append .rtf to our existing
file path
    set rtfDocFilePath to pagesDocFilePath & ".rtf"

    -- save our pages document as RTF
    tell application "Pages"
        set myDoc to open pagesDocFilePath
        save myDoc as "SLDocumentTypeRichText" in
rtfDocFilePath
        close myDoc saving no
    end tell

    -- get a result file object from KosmicTask.
    tell application "KosmicTask"
        set resultFile to result file with name "result.html"
    end tell

    -- the shell script below will expect a POSIX path
    set posixPath to POSIX path of file resultFile

    -- build an RTF to HTML convertor command
    set command to "textutil -convert html -output " & ¬
            quoted form of posixPath
    set command to command & " " & quoted form of
rtfDocFilePath

    -- do command via shell
    do shell script command

    -- return result
    return {kosmicFile:resultFile, kosmicInfo:"html file
returned"}
end KosmicTask
```

That's it, I can't take planet AppleScript any longer. I'm off!

Automation via Dark matter

The dark matter in question here is primarily Cocoa (a collection of application orientated frameworks written in Objective-C). Now, most script authors probably don't know much about bridging, a technology that allows one language to communicate with another. And there is no doubt that Cocoa and its associated supporting frameworks are complex but there is very extensive documentation available to sustain the curious. However, KosmicTask exists to make things accessible so we won't worry too much about how hard stuff can be.

The ScriptingBridge is an OS X technology for automating applications via Objective-C. JSCocoa is a bridge between JavaScript and Objective-C. Put these two things together and you have a means of automating applications from JavaScript. The following example is a duplicate of our Pages '09 to HTML file processing task:

Listing 12: Automation in JavaScriptCocoa (JSCocoa)

```
// load the ScriptingBridge framework
loadFramework("ScriptingBridge")

function kosmicTask(pagesDocFilePath)
{
```

```
    // use the ScriptingBridge framework to access the
application
    var app = SBApplication.applicationWithBundleIdentifier(
                        'com.apple.iWork.pages')

    // rtf file path
    var rtfDocFilePath = pagesDocFilePath + ".rtf"

    // open pages
    var myDoc = app.open(pagesDocFilePath)

    // save document
    myDoc.saveAs_in("SLDocumentTypeRichText", rtfDocFilePath)

    // close document
    myDoc.closeSaving_savingIn(false, null)

    // form our file result path
    // this file will be automatically deleted when the task
ends.
    var resultFile = KosmicTaskController.resultFileWithName(
                                'result.html')

    // run RTF to HTML convertor as an external process
    var args = ['-convert', 'html', '-output', resultFile,
rtfDocFilePath]
    var task = NSTask.launchedTaskWithLaunchPath_arguments(
                        '/usr/bin/textutil', args)
    task.waitUntilExit

    // return result dictionary
    return {kosmicFile: resultFile}

}
```

Application automation is often tricky due to the opacity of many application dictionaries and the ScriptingBridge has some undoubted foibles of its own but the approach is feasible. The online application help book demonstrates how to translate the above automation example into Ruby, Python, Lua, F-Script and AppleScriptObjC.

One interesting point to note in the above listing is the use of NSTask, a Cocoa foundation class for running subprocesses. Traditional JavaScript has no facility whatsoever for running external programs so in this case the bridge really extends what is possible with the language.

Conclusion++

Hopefully this brief introduction to KosmicTask has been instructive and will encourage you to explore some of the great scripting technology that is available for OS X. KosmicTask makes extensive use of plug-ins and much of the application functionality, including language support, is implemented in this way. This architecture allows developers and more ambitious users to add support for additional scripting languages and data export formats. This is a topic that we will return to in a future article.

We conclude with a final particle of code. Way back in the introduction we made a reference to scripting in C++ and the CINT interpreter used by KosmicTask was developed at CERN, where, on Earth in 2011, the hunt is on to determine whether the Higgs boson exists or not. The following task sometimes outputs an appropriate quip with regard to the reality of the elusive boson (and incidentally demonstrates how to style a complex result with

CSS by using the `kosmicStyle` result key in combination with the `kosmicData` key):

Listing 13: Fancy Higgs boson quip generation in C++

```
#include <stdlib>
#include <string>

int main(int argc, char *argv[])
{
    HiggsQuip hquip;
}

class HiggsQuip {
    public:

    HiggsQuip() {

        // quips
        string quipBoson("I told you so, says Peter, humming
contentedly.");
        string quipNoBoson("Take down that damn bunting.");

        // css
        string cssBoson("color:green; font-style:italic; font-
size:72px;");
        string cssNoBoson("color:red; font-weight:bold; font-
size:72px;");

        // simple boson detector (no big magnets required)
        srand((unsigned)time(0));
        int bosonDetector = rand() % 100 + 1;

        string quip = (bosonDetector >= 50 ? quipBoson :
quipNoBoson);
        string css = (bosonDetector >= 50 ? cssBoson :
cssNoBoson);

        // tell the world, it's ...
        printf("-- (kosmicData: '%s', kosmicStyle: '%s')",
                quip.c_str(), css.c_str());
    }

};
```

KosmicTask scans the result for the `kosmicStyle` key, interprets it as CSS and applies the defined styling to the contents of the `kosmicData` key. And today, here at least, the conclusive answer to the boson issue is:

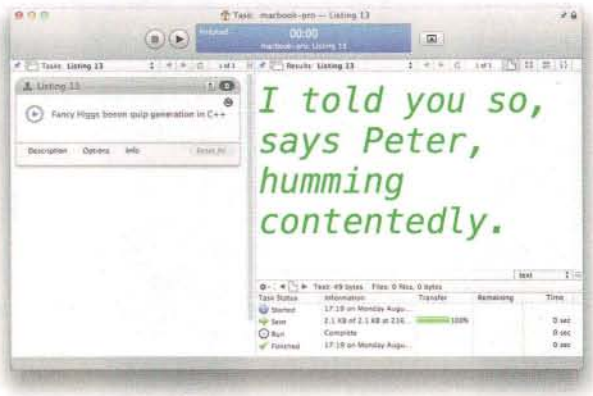


Figure 5. Task edit window displaying a complex result styled with CSS.

Web links

KosmicTask download:
http://www.mugginsoft.com/kosmictask/download
KosmicTask help book: http://www.mugginsoft.com/kosmictask/help
JSCocoa site: http://inexdo.com/JSCocoa
CINT C++ site: http://root.cern.ch/drupal/content/cint
Cocoa ref: http://developer.apple.com/technologies/mac/cocoa.html
ScriptingBridge ref:
http://developer.apple.com/library/mac/#documentation/Cocoa/Conceptual/ScriptingBridgeConcepts

About The Author

Jonathan Mitchell lives on Rathlin Island off the coast of Northern Ireland. He is the owner of Mugginsoft LLP and the developer of KosmicTask. (www.mugginsoft.com/kosmictask). With more than twenty years of independent software development experience he finds that programming for the Mac leads, via challenges and revelations, to contentment. He holds an MA in Electrical Engineering from Cambridge University and an MSc in Engineering Computation from Queen's University Belfast. You can reach him, the contented one, at jonathan@mugginsoft.com.

The Ultimate DVR for HD cable and satellite TV

eyetv

watch record edit enjoy

Award-winning EyeTV turns your Mac into the coolest television in the house. Work or surf the Web and watch TV at the same time. Record your shows to enjoy later, and play them on your iPhone® or iPad®.

eyetv HD

High Definition Video Recorder for Component Video

SRP $199.95

Elgato products are available at these and many other authorized Apple dealers:

Apple Store amazon.com MacMall OWC

elgato

WireShark and DHCP

How WireShark solved a DHCP conflict

By Mihalis & Dimitris Tsoukalos

Introduction

This is another article in the series of articles about WireShark. This article presents a real problem that was resolved using WireShark and mainly involved two DHCP servers and the DHCP protocol. The problem symptoms were very bizarre.

The network data presented here is from a simulation of the actual problem. We never publically publish actual network data due to security reasons.

The DHCP Protocol

DHCP stands for, "Dynamic Host Configuration Protocol," and is a protocol that provides configuration information to hosts on TCP/IP networks. DHCP is based on BOOTP (the *Bootstrap Protocol*) and extends it by adding more capabilities. DHCP and BOOTP protocols both use the UDP protocol with UDP ports 67 and 68.

Most of the time, DHCP provides the following information: IP address, Subnet mask, DNS servers and Default gateway although it is capable of giving many more configuration parameters.

DHCP supports, among others, the following basic messages:

DHCPDISCOVER: the client that searches for a DHCP server sends this message. It is a broadcast message (this means that it is sent to a **LAN** only using the MAC address of the client because the client does not have an IP address yet).

DHCPOFFER: when a DHCP server receives a DHCPDISCOVER message, it responds with a DHCPOFFER message.

DHCPREQUEST: The DHCPREQUEST message comes from a client, and provides information to the chosen DHCP server even if there is only one offer.

DHCPACK: this message is the response of the chosen DHCP server. It includes all the required configuration information.

Figure 1 shows a normal DHCP transaction that involves the Airport wireless card of an iMac and an ADSL router that also acts as a DHCP server.

The first packet is the DHCPDISCOVER message from the iMac searching for a DHCP server. Since the iMac does not have an IP address yet, the source IP of the packet is 0.0.0.0 and the destination IP is the broadcast IP (255.255.255.255). What distinguishes the Airport card of the iMac from the other network devices found in the same LAN is the MAC address of the Airport card, which is unique. Therefore the DHCPDISCOVER message should include the MAC address of the device requesting a DHCP server.

The next message is the DHCPOFFER from the DHCP server with IP 192.168.1.1 and is a broadcast message since the iMac still has no IP address.

Then the iMac requests from the DHCP server the offered configuration parameters with the DHCPREQUEST message. Next, the DHCP server sends a DHCPACK message back to the iMac that includes the configuration parameters. From now on, the iMac can use the offered configuration information and any

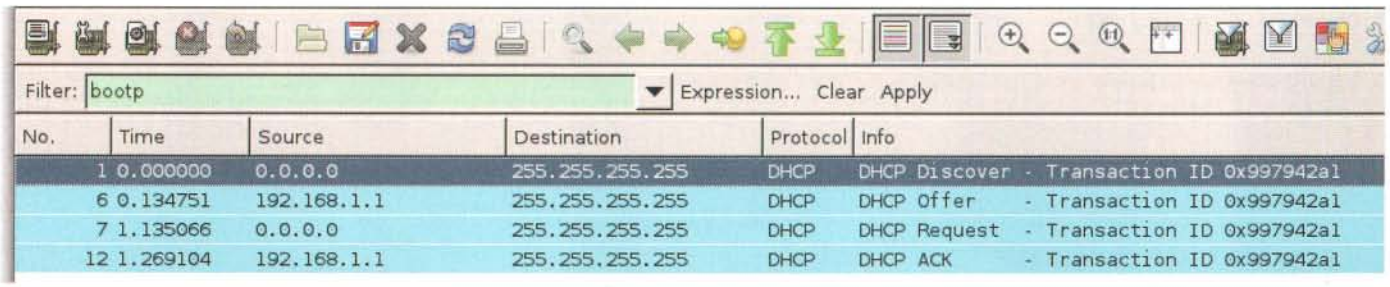


Figure 1: A usual DHCP transaction

@ScreenFlow *Because it's simply epic.*
Best Mac recording software ever.

Record. Edit. Share

ScreenFlow is powerful, easy-to-use screencasting software for the Mac. With ScreenFlow you can record the contents of your entire monitor while also capturing your video camera, microphone and your computer audio. The easy-to-use editing interface lets you creatively edit your video, add additional images, text, music and transitions for a truly professional-looking video. The finished result is a QuickTime or Windows Media movie, ready for publishing to your Web site, blog or directly to YouTube or Vimeo.

Get a free trial at www.telestream.net/screenflow

Telestream.net

	Time	Source	Destination	Protocol	Length	Info
1	0.000000	0.0.0.0	255.255.255.255	DHCP	342	DHCP Discover - Transaction ID 0x7cdcacad
2	0.007067	192.168.1.254	255.255.255.255	DHCP	321	DHCP Offer - Transaction ID 0x7cdcacad
3	0.008310	0.0.0.0	255.255.255.255	DHCP	353	DHCP Request - Transaction ID 0x7cdcacad
4	0.015050	192.168.1.254	255.255.255.255	DHCP	321	DHCP ACK - Transaction ID 0x7cdcacad
5	0.910851	10.0.10.10	255.255.255.255	DHCP	348	DHCP Offer - Transaction ID 0x7cdcacad
6	0.912131	10.0.10.10	255.255.255.255	DHCP	348	DHCP Offer - Transaction ID 0x7cdcacad

Figure 2: All DHCP-related packets

	Time	Source	Destination	Protocol	Length	Info
1	0.000000	0.0.0.0	255.255.255.255	DHCP	342	DHCP Discover - Transaction ID 0x7cdcacad
2	0.007067	192.168.1.254	255.255.255.255	DHCP	321	DHCP Offer - Transaction ID 0x7cdcacad
3	0.008310	0.0.0.0	255.255.255.255	DHCP	353	DHCP Request - Transaction ID 0x7cdcacad
4	0.015050	192.168.1.254	255.255.255.255	DHCP	321	DHCP ACK - Transaction ID 0x7cdcacad
5	0.910851	10.0.10.10	255.255.255.255	DHCP	348	DHCP Offer - Transaction ID 0x7cdcacad
6	0.912131	10.0.10.10	255.255.255.255	DHCP	348	DHCP Offer - Transaction ID 0x7cdcacad

Figure 3: The second DHCPOFFER message

	Time	Source	Destination	Protocol	Length	Info
1	0.000000	0.0.0.0	255.255.255.255	DHCP	342	DHCP Discover - Transaction ID 0x7cdcacad
2	0.007067	192.168.1.254	255.255.255.255	DHCP	321	DHCP Offer - Transaction ID 0x7cdcacad
3	0.008310	0.0.0.0	255.255.255.255	DHCP	353	DHCP Request - Transaction ID 0x7cdcacad
4	0.015050	192.168.1.254	255.255.255.255	DHCP	321	DHCP ACK - Transaction ID 0x7cdcacad
5	0.910851	10.0.10.10	255.255.255.255	DHCP	348	DHCP Offer - Transaction ID 0x7cdcacad
6	0.912131	10.0.10.10	255.255.255.255	DHCP	348	DHCP Offer - Transaction ID 0x7cdcacad

Figure 4: The first DHCPOFFER message

	Time	Source	Destination	Protocol	Length	Info
1	0.000000	0.0.0.0	255.255.255.255	DHCP	342	DHCP Discover - Transaction ID 0x7cdcacad
2	0.007067	192.168.1.254	255.255.255.255	DHCP	321	DHCP Offer - Transaction ID 0x7cdcacad
3	0.008310	0.0.0.0	255.255.255.255	DHCP	353	DHCP Request - Transaction ID 0x7cdcacad
4	0.015050	192.168.1.254	255.255.255.255	DHCP	321	DHCP ACK - Transaction ID 0x7cdcacad
5	0.910851	10.0.10.10	255.255.255.255	DHCP	348	DHCP Offer - Transaction ID 0x7cdcacad
6	0.912131	10.0.10.10	255.255.255.255	DHCP	348	DHCP Offer - Transaction ID 0x7cdcacad

Figure 5: The DHCPACK message

parameter that is unique to the iMac, like the IP address, is reserved by the DHCP server and is not offered to any other device.

The Problem

While one of us was working for a company as an administrator, the following situation came up: some computers could not connect to the company's network although my own computer was OK. After investigating the problem, I found out that after a given time no computer could correctly connect to the company's network and additionally after a properly working computer was rebooted, it could not connect to the company's network!

It is easy to guess that there was a legitimate DHCP server running on the network that supplied IP addresses and the rest of the configuration information to all computers.

Problem Analysis using WireShark

My computer was connected to an Ethernet switch using a good old 8-port Ethernet Hub. So, I connected a laptop to another port of the Hub, I started WireShark on my computer and then I switched on the laptop asking it to get its IP address using DHCP. After a few minutes, I had all the needed information to solve the problem.

Figure 2 shows all the DHCP packets that were interacting with the laptop. There were two DHCPOFFER messages from two different IP addresses (*192.168.1.254* and *10.0.10.10*) but it was supposed to be only one DHCP server (*10.0.10.10*) on the network. This was the first truly useful hint for solving the actual problem (see Figure 2).

Figure 3 (above) shows the DHCPOFFER message from the "right" DHCP server. This was the expected DHCPOFFER message. The IP address of the DHCP server is 10.0.10.10. The

SAINT® for Mac OS X

Coming soon - SAINT for Mac OS X Lion

Integrated Vulnerability Scanning, Penetration Testing, and Checklist (Benchmark) Compliance.

Vulnerability Scanning

Assess any target with an IPv4, IPv6, or URL with pre-defined policies for PCI, HIPAA, FISMA, and more.

Identify CVE, OSVDB, IAVA, OVAL, and more.

Penetration Testing

Exploit vulnerabilities to gain remote access.

Run social engineering, phishing assessments, and more with the exploit tools suite.

Checklist Compliance

Show compliance with FDCC & USGCB security configuration policies defined by NIST.

For more information–

www.saintcorporation.com/mac

1-800-596-2006

SAINT is SCAP validated by NIST & is a certified PCI approved scanning vendor (ASV).

10.0.10.10 DHCP server offered the 10.0.10.100 IP address to the laptop.

As the DHCP server did not get any answer from the client, it resent the DHCPOFFER message (packet number 6) but as you can see it was already too late (packet number 4).

Figure 4 shows the DHCPOFFER message from the "unexpected" DHCP server. The IP address of the DHCP server is 192.168.1.254. The 192.168.1.254 DHCP server offered the 192.168.1.60 IP address to the laptop. As you can guess, all computers that could not properly connect to the network were having IPs in the 192.168.1.1-253 range (See Figure 4).

In figure 5 we confirmed what we saw in figure 4: the laptop finally preferred the wrong DHCP server to get its information. The reason for choosing the 192.168.1.254 DHCP server is that it responded first. Pretty simple reason, yet it caused many problems (See Figure 5).

The Solution

After finding out that there was a second DHCP server that triggered the problem, I was able to find out on which Ethernet switch port was the physical computer that caused the problem by running some simple Cisco IOS commands–Cisco IOS is a very powerful operating system–that directed me from our core routers to the specific Ethernet switch and then to the exact switch port the second DHCP server was connected on.

This particular computer was running a virtual machine (VM). The OS on the VM had its DHCP server running and that was the cause of the problem! Pretty tricky, don't you think?

Summary

The whole DHCP problem came up one day and solved the same day in less than one hour with the help of WireShark. The problem was complicated but WireShark made everything easier.

Web links and Bibliography

WireShark site: http://www.wireshark.org/
Internetworking with TCP/IP, Volume I, Douglas E. Comer, 1991, Prentice Hall
Display Filters Reference: http://www.wireshark.org/docs/dfref/
Dynamic Host Configuration Protocol, RFC 2131: http://www.ietf.org/rfc/rfc2131.txt

About The Authors

***Mihalis Tsoukalos** enjoys digital photography, writing articles and programming his iPhone 4 and iPad. He is the author of* **Programming Dashboard Widgets**, *an eBook. You can reach him at tsoukalos@sch.gr.*

***Dimitris Tsoukalos** works as a Telecom Engineer and enjoys protocol analysis and packet inspection. You can reach him at dtsoukalos@gmail.com.*

An Awesome Email Marketing Tool for MACTECH READERS

Benchmark Email is the standard in permission-based email marketing

- ✔ List Management
- ✔ Free Newsletter Archive
- ✔ Easy to Use Drag-n-Drop Email Editor
- ✔ Powerful Personalization
- ✔ Image Gallery
- ✔ Visual Graphs for Open & Clickthrough Tracking
- ✔ Spam & Spell Checkers
- ✔ Creative & Compelling HTML Templates
- ✔ Upload Your own Template
- ✔ Easy Video Integration

MACTECH users get a 10% Lifetime Discount with this promo code 123923

Most email marketing services charge more money for less product. Not us. Benchmark Email's sophisticated suite of email marketing features lets you grow your list, send campaigns, track your data and even take online polls for an affordable price.

Plans starting at only $9.99 per month

Sign up for a FREE 30 Day Trial Today!

www.BenchmarkEmail.com

benchmark email

800.430.4095

Payload-free Packages, Part 2

Using "payload-free" packages to install software—from an extension to a full OS.

by Greg Neagle, MacEnterprise.org

MacEnterprise.org
Mac OS X enterprise deployment project

Previously in MacEnterprise

In last month's MacEnterprise column, we began a look at so-called "payload-free" packages. In traditional Apple Installer packages, the package contains a "payload' of files and directories in a compressed archive. These files and directories are extracted from the archive and copied to the target volume. Additionally, a package may contain scripts that are run before and/or after the payload is installed; these scripts often perform additional installation tasks or configuration steps.

In a "payload-free" package, the file archive is virtually empty—it may contain a single file or directory which is usually installed somewhere disposable, like `/tmp`. But the real work of the package is done in the package scripts. Payload-free packages can be used to run system configuration scripts. By wrapping the script into a payload-free package, you can include the script in package-based workflows, like building an OS installation image with InstaDMG or Apple's System Image Utility, or with DeployStudio, or with software distribution systems like Casper, Absolute Manage, or Munki.

Along those lines, last month we created a script that turned off Bluetooth, and then modified it and inserted it into a payload-free package. We could then drop that package into an InstaDMG or System Image Utility workflow and create an OS install image that ensured Bluetooth was turned off. Or we could use our favorite software distribution mechanism to install that package on existing machines, making certain Bluetooth was turned off on those as well.

Payload-free packages …with payloads?

There is another class of problem for which payload-free packages can be a possible solution: installing software that isn't distributed in the Apple package format. There are many software items that are distributed in formats that are incompatible with Apple's Installer. A common solution for enterprise deployment of this type of software is to "repackage" the software. This usually involves figuring out exactly what gets installed and then creating a new package where the payload archive contains the files and directories that are installed.

While this is often a successful technique, it can sometimes be difficult to accurately determine the complete list of files and directories to package up. More importantly, it's a time-consuming and tedious task. Worse, it must be done over and over again – for new versions; and for other items distributed in the same format.

If the software has a way to be silently installed via the command line, an alternative approach is to use a payload-free package to install the item. With a bit of care, the work you might do to create a payload-free package for this sort of software can be rapidly reused for new versions of the software or other similar items.

Installing Adobe CS5 Extensions

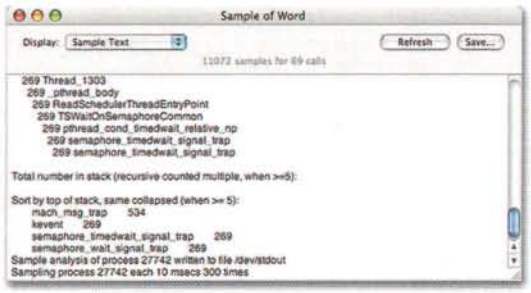
Let's consider a real-world example!

Several Adobe CS5 applications support the installation of "extensions" which add additional features and capabilities to those applications. Unfortunately, these CS5 extensions are distributed in a proprietary format and require the use of the Adobe Extension Manager CS5 application to install and remove them. But fortunately, Adobe has also documented a way to install and remove Adobe CS5 extensions via the command line. You can find that documentation here: http://macte.ch/aemcli (http://help.adobe.com/en_US/extensionmanager/cs/using/WSB4845EDD-14E5-476a-B749-FF328830D14F.html)

We can write a script that uses the command line method of installing and removing CS5 extensions, and we can wrap that script into a payload-free package. Of course, it's not really a payload-free package any longer since we are going to provide a different kind of payload. Still, the standard Apple payload at `Contents/Archive.pax.gz` will be empty.

If this sounds a bit familiar, it should. A variation of this approach is exactly how the Adobe Application Manager,

Does



+

= questions?

Are you routinely looking for answers?

Imagine a whole year of answers.

MacTech Magazine is already read every month by tens of thousands of readers. Readers that represent the very heart and soul of the Mac community. Join the crowd and sign up today!

For a special one year subscription, visit:

store.mactech.com

MACTECH®

Toll Free 877-MACTECH, Outside US/Canada: 805-494-9797

Enterprise Edition (AAMEE) creates Apple installation packages from Adobe installation media. AAMEE-generated packages are payload-free packages that run the Adobe installer tools as package scripts.

For our example, we'll use a Photoshop CS5 extension known as "John's Artists Brushes". You can read about them on John Derry's website here: http://www.johnsartistsbrushes.com/. Feel free to substitute any CS5 extension, however, if you are following along.

Like many other Photoshop CS5 extension, John's Artists Brushes are distributed in a ".zxp" archive file. For individual users, the user can just double-click on the "John_Derry_Installer.zxp" file. Adobe Extension Manager CS5 will launch and walk the user through installing the extension. See Figure 1 for an example.

Figure 1 – Adobe Extension Manager CS5

What works well for the individual user is a pain for the system administrator. Any self-respecting system administrator does not want to have to walk around double-clicking icons and clicking buttons and agreeing to licenses. The command line to the rescue! Adobe CS5 extensions can be installed by calling Adobe Extension Manager CS5 from the command line with various options. We can install John's Artist's Brushes like so:

```
sudo /Applications/Adobe\ Extension\ Manager\ CS5/Adobe\
Extension\ Manager\ CS5.app/Contents/MacOS/Adobe\ Extension\
Manager\ CS5 -suppress -install zxp="John_Derry_Installer.zxp"
```

(This is all one very long line.) We call the Adobe Extension Manager CS5 executable; tell it to suppress the license dialog box and any success confirmation, and tell it to install the "zxp" archive named "John_Derry_Installer.zxp".

Once we've verified that we can install this extension via the command-line, we can create a payload-free package that does it as well. Like last month, we can start with the payload-free package template available here: http://goo.gl/5bRoR (http://dl.dropbox.com/u/8119814/payload-free.pkg.zip)

You can download a working postflight script to add to the package template, in `Contents/Resources` from the MacTech source code archive at ftp://ftp.mactech.com.

This Python script is probably a bit more complex than you expected. It turns out that there is a complication: Adobe Extension Manager CS5 is an application that presents a user interface—even if we call it via command-line it wants access to the window manager. This is a problem if we try to do this when

Your business at your fingertips.

With Daylite for iPad you'll have access to your critical business data no matter where you are, letting you make important business decisions even when you and your employees are on different sides on the world.

Daylite

Visit www.marketcircle.com/daylite for a free 30-day trial.

the machine is at the loginwindow. If we don't take special action when calling Adobe Extension Manager when in this state, the postflight script will fail and you'll see errors similar to these in `/var/log/installer.log`:

```
installer[573]: ./postflight: Adobe Extension Manager CS5[578]
<Warning>: 3891612: (connectAndCheck) Untrusted apps are not
allowed to connect to or launch Window Server before login.
installer[573]: ./postflight: Adobe Extension Manager CS5[578]
<Error>: kCGErrorFailure: Set a breakpoint @
CGErrorBreakpoint() to catch errors as they are logged.
installer[573]: ./postflight: _RegisterApplication(), FAILED
TO establish the default connection to the WindowServer,
_CGSDefaultConnection() is NULL.
```

To avoid this issue, we need to indirectly call Adobe Extension Manager using `` `launchctl bsexec` `` so that Adobe Extension Manager runs in the same context as the loginwindow. All of this logic is encapsulated in the `runCommand(cmd)` function, which calls `getconsoleuser()` to determine if we are at the loginwindow, and `getPID(processname)` to get the process id of the loginwindow. By using `` `launchctl bsexec` `` to call it indirectly, Adobe Extension Manager will successfully install extensions when no-one is logged in, or the machine is switched to the loginwindow in a Fast User Switching environment.

Let's walk through the main part of the script, the beginning of which is marked `#### main ####`. First, we get the arguments passed to us by the installer. The path to the package itself is passed in `sys.argv[1]` (`$1` if this was a shell script), and the path to the target volume is in `sys.argv[3]` (`$3` if this was a shell script).

If the target volume is not the startup volume, we print an error message and exit. I haven't tested to see if Adobe Extension Manager installs extensions correctly when it is on volume other than the startup disk, so this script doesn't even try in this situation.

Next, we look for the Adobe Extension Manager CS5 binary, and again print an error message and exit if we can't find it.

Now, finally, we arrive at the meat of the script. Instead of hard-coding the name of the CS5 extension to install, instead the script looks through all the files in the package's `Contents/Resources` directory, looking for filenames ending with ".zxp" and ".mxp". For any files that match, it then calls the Adobe Extension Manager CS5 binary to install them.

You can see that you'll need to copy the CS5 extension to the `Contents/Resources` directory, since that's where the script looks for CS5 extensions to install. The useful thing about this approach is that it is quickly reusable – you can install other CS5 extensions by making a copy of this package and replacing the CS5 extension(s) in `Content/Resources` with other CS5 extensions.

A working version of the payload-free package will look something like Figure 2. You also will probably want to edit the `Info.plist` and `Description.plist` files to reflect the current name and description of the package. See last month's MacEnterprise column for details.

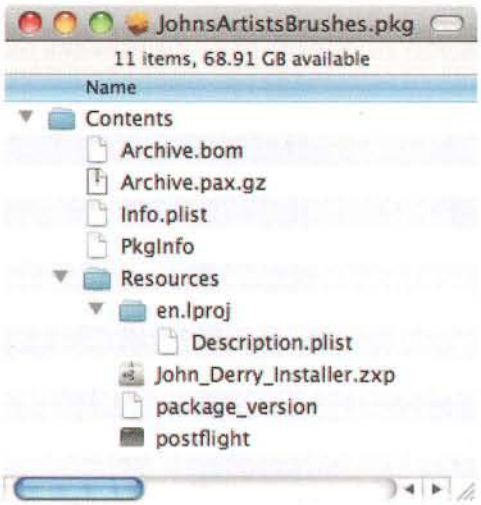


Figure 2 – A completed JohnsArtistsBrushes.pkg

You can find a completed version of this package (minus the actual CS5 extension) here: http://goo.gl/fDjQv (http://dl.dropbox.com/u/8119814/CS5_extension_template.pkg.zip).

Installing Lion

Installing a CS5 extension is a pretty small task. Adobe Application Manager, Enterprise Edition (AAMEE) generates payload-free packages that can install the entire Adobe CS5 Master Collection, complete with all current updates. But we can think even bigger. It is possible to install (or upgrade to) Mac OS X Lion via a payload-free package!

Prior to the release of Mac OS X Lion, installing a major OS upgrade always involved booting from an alternate disk. This might be a DVD, an external disk, or a network-based NetBoot disk. Since prior major OS releases came on physical DVDs, this worked well enough. But with Lion, Apple's preferred distribution method is the Mac App Store. Mac owners can buy Lion, which is downloaded as an "Install Mac OS X Lion" application to their `/Applications` folder. The Mac owner then can upgrade to Lion by running the application. The "Install Mac OS X Lion" application can even install Lion on the current startup disk, a new trick for Mac OS X installers.

In an enterprise environment, upgrading existing machines using the "Install Mac OS X Lion" application is not an ideal approach. You'd either have to have a technician visit each machine and manually download and run the application, or figure out some way to have each user do this set of tasks.

Fortunately, you can still use the same techniques for deploying Lion that you could use with previous releases of Mac OS X. It is possible to install Lion using Apple's NetInstall or NetRestore, or using a third-party tool such as DeployStudio. All of these methods can use the `InstallESD.dmg` disk image that is inside the "Install Mac OS X Lion" application bundle as a replacement for a physical OS X install DVD that you might have

used for previous releases. Some of these methods can be automated to some degree as well.

But Lion also opens up a new deployment option. Since the "Install Mac OS X Lion" application can install Lion on the current startup disk, it seems like it should be possible to use a script or payload-free package to perform the same prep work that the "Install Mac OS X Lion" application does in order to perform the install.

It turns out that it is indeed possible. The "InstallLion.pkg" tools, available at http://code.google.com/p/munki/downloads/list, can help you create a package that you can use to install Lion.

You can use this package with any deployment tool that utilizes Apple packages: Munki, Casper or Absolute Manage, as examples. You can even use this package with Apple Remote Desktop and DeployStudio.

What you need

To create and use a payload-free package to install Lion, you need three things:

The "InstallLion.pkg" tools. They are available as a zip archive at the above URL, or in a Git repository you can clone using: `git clone https://code.google.com/p/munki.installlionpkg/`

A copy of the "Install Mac OS X Lion" application, downloaded from the Mac App Store, or a copy of the `InstallESD.dmg` for a specific hardware model obtained with the methods described here: http://www.afp548.com/article.php?story=getting-lion-installers

A license for Mac OS X Lion for each machine to which you will deploy Lion.

What you do

Once you've downloaded and expanded the zip archive, or cloned the Git repo, you'll have a set of files something like Figure 3.

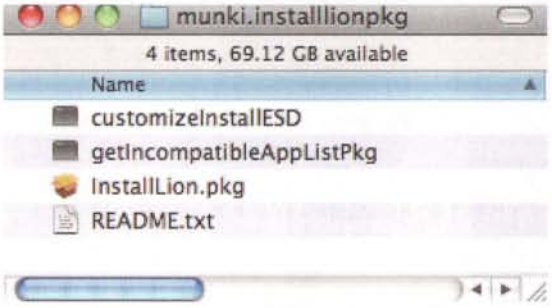


Figure 3 – InstallLion package tools

Take some time to review the `README.txt` file—it will have up to date information, which may have changed a bit since this article was submitted for publication.

For a basic install of Lion, you only need to copy the `InstallESD.dmg` file into the `Content/Resources` directory of `InstallLion.pkg`. The `InstallESD.dmg` can be found

inside the "Install Mac OS X Lion" application inside the `Contents/SharedSupport` subfolder.

(If you are working in the Finder, you can show the contents of both the "Install Mac OS X Lion" application and the `InstallLion` package by control-clicking on each and choosing Show Package Contents from the popup menu. You can then option-drag the `InstallESD.dmg` into the `InstallLion` package. See Figure 4.)

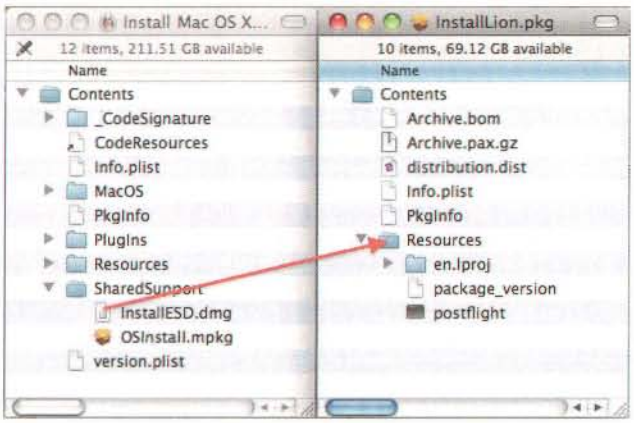


Figure 4 – Copying InstallESD.dmg to InstallLion.pkg

That's all you need to do for a basic non-customized installation – `InstallLion.pkg` can now be used to do an unattended install of Mac OS X Lion. Use your favorite software distribution method to install this package on machines you wish to upgrade to Lion.

How it works

`InstallLion.pkg` is a "payload-free" package with a dummy payload. The real work is done in a package postflight script located at `InstallLion.pkg/Contents/Resources/postflight`. It's written in Python, so you can examine it at detail if you wish.

The postflight script performs an approximation of the actions that the GUI "Install Mac OS X Lion" application performs when you choose to install Lion on the current startup disk.

Those actions are:

Create a "Mac OS X Install Data" directory at the root of the target volume.

Mount the `InstallESD.dmg` disk image.

Copy the `kernelcache` and `boot.efi` files from the disk image to the "Mac OS X Install Data" directory.

Unmount (eject) the `InstallESD.dmg` disk image.

If the `InstallLion.pkg` is on the same volume as the target volume, create a hard link to the `InstallESD.dmg` disk image in "Mac OS X Install Data", otherwise copy the `InstallESD.dmg` disk image to that directory.

Create a `com.apple.Boot.plist` file in the "Mac OS X Install Data" directory. This tells the kernel how to mount the disk image to use for booting.

Create a `minstallconfig.xml` file, which tells the OS X Installer what to install and to which volume to install it. It also

provides a path to a `MacOSXInstaller.choiceChanges` file if one has been included in the package.

Create an `index.sproduct` file and an `OSInstallAttr.plist` in the "Mac OS X Install Data" directory. These are also used by the OS X Installer.

Set a variable in nvram that the OS X Installer uses to find the product install information after reboot.

Use the `bless` command to cause the Mac to boot from the kernel files copied to the "Mac OS X Install Data" directory.

The next step would be to reboot, but the postflight script does not do this; it just exits. The package is marked as requiring a reboot, so whatever mechanism is used to install the package is responsible for rebooting as soon as possible after the install.

Upon reboot, the machine boots and runs the Lion Installer just as if you had run the "Install Mac OS X Lion" application manually. It creates a "Recovery HD" partition if needed and possible, and then installs Lion on the target volume, displaying the OS X Installer GUI. When installation is complete, the machine reboots a second time, this time booting from the new Lion installation.

Customizing the install

You can customize the Lion install by providing a `MacOSXInstaller.choiceChanges` file, and you can install additional packages after the installation by adding them to the `InstallESD.dmg` file using the `customizeInstallESD` tool. See the `README.txt` for details on these customizations.

Conclusion

We've seen that payload-free packages are a very useful thing for a Mac OS X administrator to have in his or her tool belt. They can be used to run system configuration scripts, to add or remove printer queues and to remove previously installed software.

Since payload-free packages are essentially a way to run a script, they can also be used to install software – and are useful for installing software that is not distributed in Apple package format. If you can install the software via a script or a tool that can be called via the command-line, instead of repackaging the software, you may be able to create a payload-free package that installs the software using the vendor-supported tools.

MT

About The Author

Greg Neagle is a member of MacEnterprise (macenterprise.org) and is a senior systems engineer at a large animation studio. Greg has been working with the Mac since 1984, and with Mac OS X since its release. Greg Neagle and Edward Marzcak's book: Enterprise Mac Managed Preferences, which covers Apple's Managed Preferences, was recently published by Apress. Greg can be reached at gregneagle@mac.com.

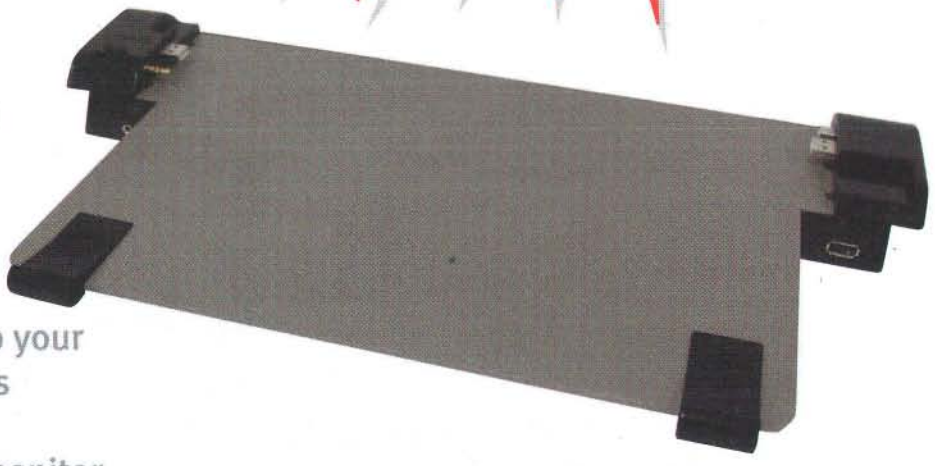
BookEndz®

Docking Stations for Apple Computers

DOCKING STATIONS FOR APPLE COMPUTERS

Turns your laptop into a desktop

NEW!
MacBook Air
11" and 13"
Docking stations



- Includes USB hub to give you a total of 4 USB 2.0 ports
- Allows access to SD slot
- Adds an Ethernet Connection to your MacBook Air for wired networks
- Allows you to use an external monitor in addition to your laptop
- Angles your MacBook Air off a flat surface

Visit our website for latest product announcement www.BookEndzdocks.com

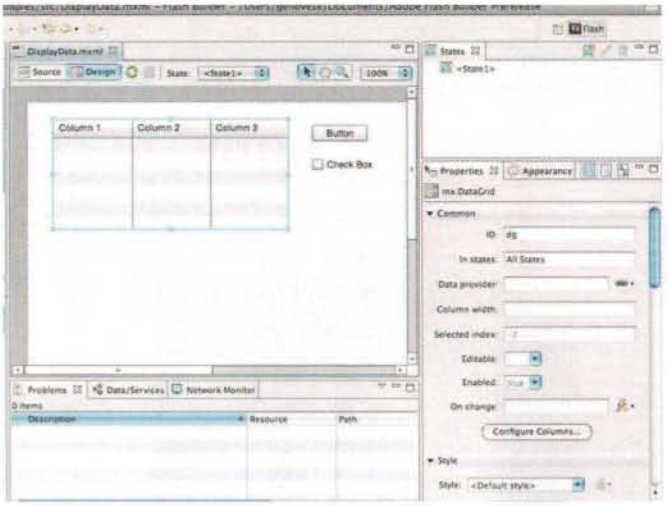
BookEndz®

Manufactured by Olympic Controls
1250 Crispin Drive • Elgin, Illinois 60123 • USA

Phone: 847-742-3566 • Fax: 847-742-5686 • Toll Free: 888-622-1199 • E-mail: Sales@BookEndzdocks.com

Adobe Flash Builder and Flex

by Dennis Sellers



Adobe's (http://www.adobe.com) Flash Builder 4.5 and Flex 4.5.1 software enables developers to build applications for iPhone, iPad and BlackBerry PlayBook devices. Support for Android was released in April 2011.

"Developers have a single platform for building highly expressive mobile applications and can be distributed via the Apple App Store, Android Marketplace, and BlackBerry App World. Offered standalone or as part of Creative Suite 5.5 Web Premium and Master Collection, Flash Builder 4.5 enables the creation of applications that work seamlessly across leading mobile device platforms," says Ed Rowe, vice president of developer tooling, Adobe.

Adobe has also recently announced the Adobe Digital Enterprise Platform. With consumers increasingly engaging with more content on smartphones and tablets, large enterprises are required to add mobile applications as a core part of their marketing strategy. Simultaneously, IT organizations are looking to build business process, CRM and other employee productivity applications to support the broad range of devices and platforms in use today. Using Flash Builder and Flex, enterprises can now use one tool, one framework and one codebase to create high performance applications that run on desktops as well as on smartphones and tablets, says Rowe.

Flex 4.5.1 also includes productivity enhancements that allow developers to build applications for the desktop, Web and top mobile platforms. Using best practice code templates, code completion and code generation features in Flash Builder 4.5, developers can accelerate the creation of Flex and ActionScript applications and deploy them using Adobe AIR software, Adobe's runtime for standalone applications.

Adobe is showing off a number of mobile applications built using Flash Builder 4.5 and Flex 4.5.1:

- Conqu (http://adobe.ly/m46BTl) is a task management tool designed to help conquer an email inbox and get things done;
- Mr. Mixit (http://adobe.ly/meeMSk) is a spin-based matching game where users must mix record labels against the clock.
- Muni Tracker (http://adobe.ly/ihOnUp) tracks San Franisco Muni locations and arrival time predictions on a map updating live every 10seconds. Favorite transit stops and lines can also be bookmarked for easy access later.
- PolitiFact Mobile App (http://adobe.ly/ky4GqF) helps you "find the truth in politics," with features including the Truth Index, Truth-O-Meter and Flip-O-Meter.
- Pyramix (http://adobe.ly/meeMSk) is a word game that combines the strategy of Cryptoquote with the simplicity of Boggle.

For iOS, developers are not writing an Objective-C app, but according to Adobe, the byte code generated looks very similar to those apps. Because the tools are cross platform, however, new features introduced in iOS will lag behind native Objective-C applications. Developers targeting the richest iOS or other platform experience or need specific environment features, should probably use native tools. That said, for those that need cross platform most, these tools give you that core benefit.

To purchase or download a trial version of Flash Builder 4.5, go to http://www.adobe.com/products/flash-builder.html . The street price is US$249 for Flash Builder 4.5 Standard, $699 for Flash Builder 4.5 Premium. Flash Builder 4.5 Premium is also available as part of the Adobe Creative Suite 5.5 Web Premium and Master Collection. Upgrade pricing for Flash Builder 4.5 is $49 and volume licensing is available. Flex 4.5.1 is available as a free open source framework. Also, Flash Builder for PHP now supports mobile application.

MT

About The Author

Dennis Sellers is a long time journalist. He started in the newspaper business, but has been in the online journalism business for the past 15 years. He's the editor/publisher of Macsimum News (http://www.macsimumnews.com)

Advertiser/Product Index

Ad Index by Company for: MacTech Mag 27.09 (September/2011)

Ad Index by Product for: MacTech Mag 27.09 (September/2011)

THE MACTECH SPOTLIGHT

HISHAM KHALIFA

Binary Bakery – http://www.binarybakery.com

What do you do?

By day I work as a consultant for a European investment bank dealing with mergers and acquisitions and fundraising—so you can imagine it's as far removed from coding as can be. By night, I burn the midnight oil working on what I'm really passionate about; programming in C and Objective-C (with a bit of assembly sometimes thrown in). As is typical of one-man indie outfits, I'm the chief baker, the chief support officer, the chief everything with an awesome wife who supports me fully as I toil away developing apps in unsocial hours.

How long have you been doing what you do?

I've always been interested in programming since I was a kid. I think I was around six when I typed in my first program in BASIC. My dad also enrolled me in computer courses during that time.

As for Binary Bakery, it all started a couple of years back when I was between jobs and I had about three months of time to kill. I also had just finished reading Aaron Hillegass' excellent *Cocoa Programming for Mac OS X* and was itching to write an app. The right idea came along as I was working in Photoshop on my dual-display setup and was frustrated with Mac OS X's somewhat lacking multi monitor support; this prompted me to develop MenuEverywhere.

I never intended to commercialize the app, but after having worked on it for some time, I thought it wouldn't hurt to put it up as shareware; a sort of litmus test to see if others would actually pay for something I wrote. Within ten minutes of it being listed in one of the Mac app directories I sold my first license to some guy in Germany.

What was your first computer?

A VTech Apple II clone; it was the first machine that I played around with. Soon after we got a Commodore 64 and that's the machine on which I took my first real steps to learn programming.

What's the coolest tech thing you've done using OS X?

MenuEverywhere - an app that recreates the menu bar on any screen and on any window. When writing a user interface enhancement app that's supposed to behave well with virtually all other Mac OS X apps ever written, there are lots of variables that could go wrong. I'm quite pleased with the way MenuEverywhere has developed and matured to the point of working well with almost all apps out there.

Ever?

Ever? I'd say writing my own Accessibility API framework that's useful for coding all the app ideas I have for everything from window management to menu management.

What is the advice you'd give to someone trying to get into this line of work today?

Perseverance, discipline and passion are musts for indie devs. Developing on your own there is the risk of ending up doing something that may or may not end up paying you back (be it user satisfaction or sales) for the time commitment. Learn from your mistakes, tap into the wonderful Mac OS X dev community and keep at it.

Anything that you consider indispensable for your work?

My library of development books, everything from Kernighan & Ritchie's C book to Hillegass - without such books I'd have no direction to learn what I've learnt, especially since I have no formal computer science training.

Where can we see a sample of your work?

Trial versions of my apps are available at www.binarybakery.com

The next way I'm going to impact the Mac universe is:

There are a few neat user interface paradigms out there that have either been left out or deprecated from OS X. I'm working on the next major version of MenuEverywhere which will have some pretty interesting approaches to "menuing" and selecting commands.

Anything else we should know?

There aren't really any barriers or obstacles to entry in terms of indie development. It's all about commitment and taking the time to learn as much as you can. Look at me; I live all the way in the tiny island state of Bahrain where you either end up working in the financial sector or tourism sector. There is no tech sector.

That hasn't stopped me from learning programming and writing apps that make thousands of Mac users that little bit more pleased when using their machines.

If you or someone you know belongs in the MacTech Spotlight, let us know! Send details to editorial@mactech.com

Mac shopping made easy.

Grab that to-do list, and prepare for some one-stop shopping at Smalldog.com!

Bundles simplify the buying process

Mac bundles (think Mac + RAM + AppleCare + external hard drive, etc.) not only include **everything you need,** but also **save you money.**

Visit » **Smalldog.com/specials**

Macs from under $500

We carry all **current Macs** as well as **used, refurbished and closeout models,** so there is a Mac for any budget.

Visit » **Smalldog.com/macs**

Free shipping over $200

It's true–we provide **free, same-day ground shipping** on every item over $200 every day.

Tax-free shopping

Purchases outside of Vermont are always shipped **tax-free.**

✓ 13" MacBook Pro + Chill Pill® mobile speakers

Small Dog Electronics

Always By Your Side

www.smalldog.com

800-511-MACS

Apple Specialist

Celebrating 15 Years • 3rd Largest Apple Specialist in New England • 5-Star Merchant Rating • Same-day shipping

Bundles

Macs

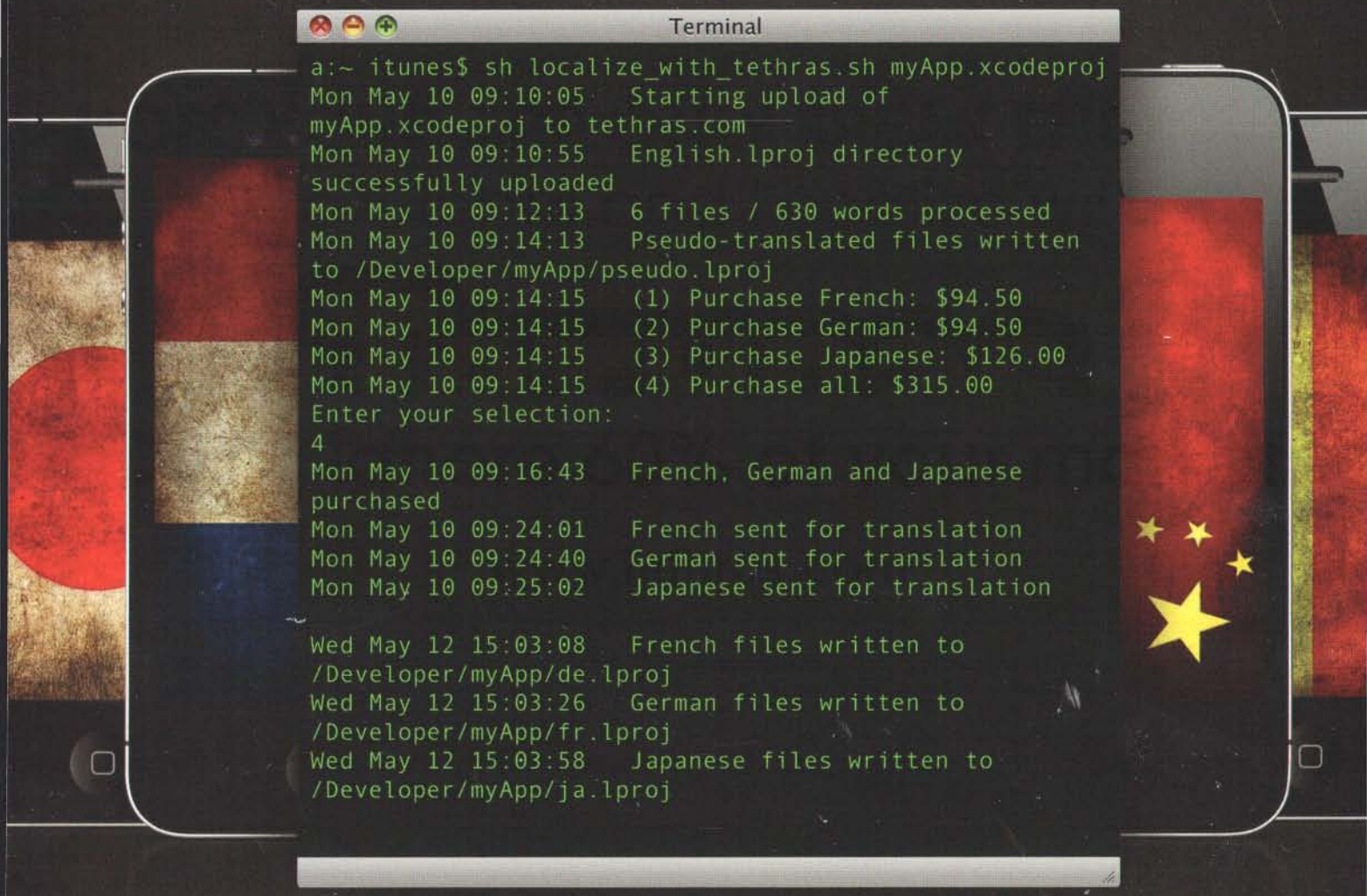
Free Shipping

Tax-Free

DOES YOUR APP ONLY SPEAK ENGLISH?

Go Global and Localize with

TETHRAS



Don't ignore 60% of your market

www.tethras.com